НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри
_____ Сергій СТІРЕНКО

“___”_____2020 р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп’ютерні системи та мережі»

спеціальності 123 «Комп’ютерна інженерія»

на тему: «Програмно-апаратний комплекс для вирішення задач лінійної алгебри»

Виконав:
студент IV курсу, групи ІО-62
Іван ОСІПОВ _____

Керівник:
Доцент, к.т.н.,
Олександр КОРОЧКІН _____

Консультант з нормоконтролю:
Професор, д.т.н.,
Валерій СІМОНЕНКО _____

Рецензент:
Доц. каф. СПКС, к. т. н., доц.
Оксана ТАРАСЕНКО-КЛЯТЧЕНКО _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврьский)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___» _____2020 р.

ЗАВДАННЯ
на дипломний проект студента

Осіпова Івана Олеговича

1. Тема проекту «Програмно-апаратний комплекс для вирішення задач лінійної алгебри»_____
керівник проекту Корочкін Олександр Володимирович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «07» травня 2020 року №1081-с
2. Термін здачі студентом закінченого проекту
_____
3. Вихідні дані до проекту *технічна документація*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Створення програмно-апаратного комплексу для вирішення задач лінійної алгебри на основі існуючих рішень. Опис предметної області, дослідження засобів розробки програмного забезпечення, розробка алгоритму для вирішення задач лінійної алгебри програмна реалізація та тестування.
5. Перелік графічного матеріалу
Принципова схема, функціональна схема та структурна схема_____

6. Консультанти проекту, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
нормоконтроль	д.т.н., проф. Сімоненко В.П.		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту	Строк виконання етапів проекту	Примітки
1.	*Затвердження теми роботи*	*1.09.2019*	
2.	*Вивчення та аналіз завдання*	*2.09.2019-12.03.2020*	
3.	*Розробка архітектури та загальної структури програми*	*12.03.2020-22.03.2020*	
4.	*Розробка структур окремих Інтерфейсів програми*	*22.03.2020-2.04.2020*	
5.	*Програмна реалізація*	*2.04.2020-13.04.2020*	
6.	*Оформлення пояснювальної записки*	*13.04.2020-21.05.2020*	
7.	*Захист програмного продукту*	*21.05.2020 – 25.05.2020*	
8.	*Передзахист*	*26.05.2020*	
9.	*Захист*		

Студент Іван ОСІПОВ

Керівник Олександр КОРОЧКІН

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1.	А4		Завдання на дипломний проект	2	
2.	А4	ДП.4665.02.000 ТЗ	Технічне завдання	3	
3.	А4	ДП.4665.03.000 ПЗ	Пояснювальна записка	50	
4.	А4	ДП.4665.04.000 А1	Принципова схема алгоритму	1	
5.	А4	ДП.4665.05.000 А2	Функціональна схема	1	
6.	А4	ДП.4665.06.000 А3	Структурна схема	1	

					ДП.4665.01.000 ВП			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Осіпов І.О.			Програмно-апаратний комплекс для вирішення задач лінійної алгебри Відомість дипломного проекту	Літ.	Аркуш	Аркушів
Перевірив		Корочкін О.В.					1	1
Реценз.						НТУУ «КПІ», ФІОТ, ІО-62		
Н. Контр.		Сімоненко В.П.						
Затв.								

Технічне завдання до дипломного проекту

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на створення паралельної програми для зібраного апаратного комплексу.

Область застосування: створення програми під апаратний комплекс для вирішення задач лінійної алгебри.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить завдання на виконання розробки програмно-апаратного комплексу для вирішення задач лінійної алгебри, затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний Інститут».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є створення програми для вирішення задач лінійної алгебри під створену систему на основі існуючих рішень.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки служать науково-технічна література з комп’ютерних технологій, публікації в періодичних виданнях, довідники з програмування, публікації в Інтернеті за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Створення апаратної частини на основі існуючих рішень
- Розробка програмного забезпечення для вирішення задач лінійної алгебри
- Виконання програмної емуляції роботи та збір результатів.

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows XP, MS Windows Vista, MS Windows 7, MS Windows 8/8.1, MS Windows 10
- Java SE 7 і вище

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel i7 і вище
- Оперативної пам’яті не менше 2 Гбайт

Анотація

Робота присвячена створення програмно-апаратного комплексу для вирішення задач лінійної алгебри. Через зростання актуальності багатопотокових обчислювальних системи, та великим обсягом обчислень лінійних рівнянь у різних сферах діяльності.

Запропонований метод полягає у створенні комп'ютерної системи на основі існуючих комплектуючих, та написанні програмного забезпечення для вирішення задач лінійної алгебри, яка буде максимально ефективно вирішувати задачі за допомогою багатопотоковості. Це дозволить суттєво зменшити час виконання.

Abstract

The work is devoted to the creation of a software and hardware complex for solving problems of linear algebra. Due to the growing relevance of multithreaded computing systems, and the large volume of calculations of linear equations in various fields.

The proposed method is to create a computer system based on existing components, and write software to solve problems of linear algebra, which will most effectively solve problems using multithreading. This will significantly reduce execution time.

Пояснювальна записка
до дипломного проекту
на тему: «Програмно-апаратний комплекс для вирішення задач лінійної алгебри»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

ПАК	Програмно-апаратний комплекс.
CPU	(англ.Central Processing Unit) Центральний процесор
GPU	(англ.Graphics Processing Unit) Графічний процесор.
API	(англ. Application Programming Interface) набір визначень взаємодії різнотипного програмного забезпечення.

ВСТУП

Розрахункові задачі, які виникають при математичному моделюванні в галузях науки та інженерії, часто являються задачами лінійної алгебри. Як правило розв’язування задач лінійної алгебри займає значну частину часу розв'язання всієї задачі в цілому.

При пошуку інформації для свого диплому я зрозумів, що особливістю задач лінійної алгебри найчастіше є високий порядок матриць, що потребує високої продуктивності в обчислювальних процесах

Для цього потрібно великі обчислювальні ресурси та програмне забезпечення.

Саме тому метою моєї роботи є створення ПАК для обчислення задач лінійної алгебри, що дозволить оптимізувати час виконання:

Для того щоб досягнути мети потрібно зробити наступне:

- Провести аналіз апаратного забезпечення;
- Обрати апаратне забезпечення;
- Провести аналіз програмного забезпечення;
- Обрати програмне забезпечення;
- Створити програму;
- Протестувати зібраний ПАК.

РОЗДІЛ 1. АПАРАТНІ ЗАСОБИ ПОБУДОВИ ВИСОКОПРОДУКТИВНИХ КОМПЛЕКСІВ

У наше століття бурхливого розвитку цифрових технологій намічаються дві основні тенденції: з одного боку, доступ до інформації спрощується і для її обробки не потрібні значної обчислювальної потужності, наприклад, для спілкування і соціальних сервісів, а також споживання медіа контенту цілком достатньо можливостей смартфона або планшета, але з іншого боку, фото реалістичні відеоігри, редагування відео з роздільною здатністю 4К і, особливо, моделювання віртуальної реальності вимагають продуктивного апаратного забезпечення. Саме такі ресурсо місткі завдання є драйверами, що рухають вперед ринок комп'ютерних комплектуючих і на передньому краї, безперечно, знаходяться графічні акселератори, а також центральні процесори. Втім, якщо в сегменті відеокарт протягом ось уже багатьох років триває запекле протистояння компаній Advanced Micro Devices і NVIDIA, то на ринку CPU з архітектурою x86 боротьба між двома найбільшим Чіпмейкер - Intel і AMD - спостерігається тільки в молодшому і середньому цінових діапазонах.[1]

1.1 Високопродуктивні процесори Intel

1.1.1 Процесори дев’ятого покоління

Платформа LGA2066 і процесори сімейства Skylake-X були представлені Intel в 2017 році. Спочатку це рішення націлювалось компанією на сегмент HEDT, тобто на високопродуктивні системи для користувачів, які займаються створенням і обробкою контенту, адже Skylake-X містили істотно більшу кількість обчислювальних ядер в порівнянні зі звичними представниками сімейств Kaby Lake і Coffee Lake.

Однак за час, що минув з появи Skylake-X, ландшафт на процесорному ринку істотно змінився, і сьогодні досить доступні CPU можуть мати шість або навіть вісім обчислювальних ядер, а перспективні масові CPU можуть отримати десять або навіть дванадцять ядер. Чи робить це Skylake-X марними чіпами? Скоріше за все ні. По-перше, серед представників цієї серії існують пропозиції з 16 і 18 ядрами, і подібних

до них масових варіантів найближчим часом на ринку точно не буде. По-друге, в платформі LGA2066 закладені й інші переваги, які відрізняють їх від звичайних споживчих процесорів, наприклад перевагу в числі каналів пам'яті і доступних ліній PCI Express.[2]

Тому косметичне оновлення модельного ряду Skylake-X, яке мікропроцесорний гігант провів в кінці минулого року, здалося цілком закономірним - воно прекрасно вписувалося в прийнятий у Intel графік щорічних анонсів. Однак ставлення виробника до своїх HEDT-новинок трохи здивувало: компанія не тільки не стала переглядати ціни, але і до того ж відмовилася надавати зразки процесорів IT-пресі, обмежившись лише формальної презентацією і подальшим початком продажів.[3]

Найпомітнішою зміною в нових процесорах порівнянно з попередніми моделями Skylake-X семитисячної серії, стало зростання тактових частот. Номінальні частоти піднялися на 200-600 МГц, а максимальні частоти, що досягаються при включенні турборежиму, виросли на 200-300 МГц. Крім того, у молодших представників серії збільшився обсяг кеш-пам'яті третього рівня. Раніше він розраховувався виходячи з правила «1,375 Мбайт на ядро», а тепер на кожне ядро може припадати приблизно до 2 Мбайт кеша. І останнє: контролер PCI Express восьмиядерного Core i7-9800X був повністю розблоковано, завдяки чому цей процесор отримав в своє розпорядження всі 44 лінії, які раніше були доступні тільки в процесорах з числом ядер 10 і більше.[4][5]

Однак всі ці приємні зміни спричинили за собою і зростання тепловиділення. У той час як перше покоління Skylake-X мало теплової пакет, обмежений рамками 140 Вт, в нових процесорах характеристика TDP збільшена до 165 Вт. Іншими словами, за підвищені частот, які присвоєні новим процесорам без будь-яких принципових змін в вживаному для їх випуску 14-нм технологічному процесі, доводиться розплачуватися розширеними енергетичними та тепловими межами. Правда, сама Intel при цьому стверджує, що підняти швидкісні характеристики дозволило впровадження виробничої технології третьої версії з умовною назвою 14 ++ нм, по якій зараз виготовляються процесори Coffee Lake і Coffee Lake Refresh. І якби не це, то тепловиділення могло б бути ще вище. Але побоюватися, що нові

Skylake-X можуть бути схильні до перегріву, причин немає. Знизити робочі температури нових процесорів повинен покращений термоінтерфейсний матеріал під теплорозподільної кришкою. Місце яку застосовували раніше полімерної термопасти зайняв припій із заздалегідь більш високою теплопровідністю.

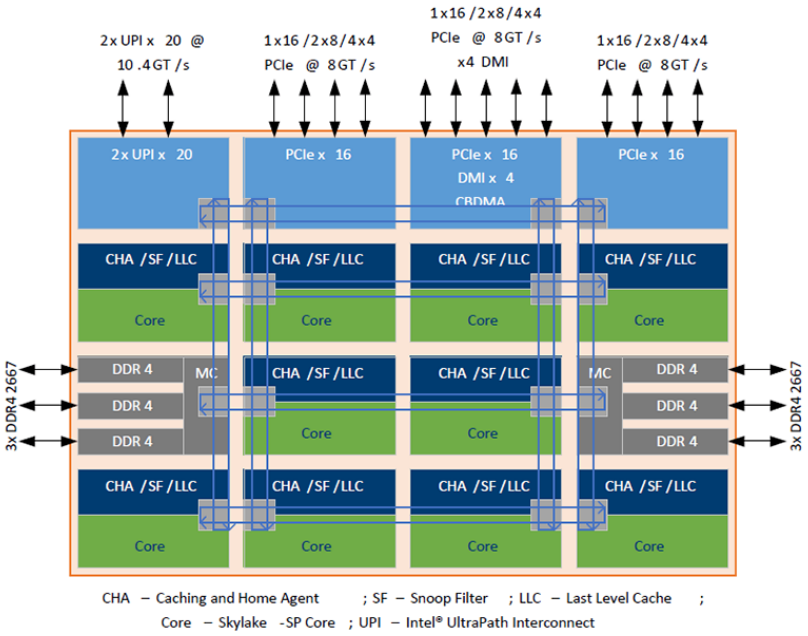


Рисунок 1.1 - архітектура процесорів intel дев’ятого покоління

Всі нові HEDT-процесори дев'ятитисячний серії, в тому числі і восьми- і десятіядерние варіанти, базуються на кристалі HCC. Тобто навіть в Core i7-9800X або Core i9-9900X потенційно є 18 ядер, але значна їх частина апаратно заблокована на стадії виробництва.[6]

Таке дивне на перший погляд рішення було прийнято саме заради збільшення обсягу кеш-пам'яті в нових процесорах. Внутрішній устрій Skylake-X припускає, що на кожне обчислювальне ядро виділяється порція кеш-пам'яті об'ємом 1,375 Мбайт. І якби в тому ж Core i9-9900X використовувався молодший кристал LCC, більш 13,75 Мбайт L3-кеша цей процесор завідомо отримати б не міг. Більший кристал HCC в цьому плані більш гнучкими, в ньому в цілому закладено 24,75 Мбайт кеша, і цей збільшений обсяг частково задіяний в восьми- і десятіядерних процесорах нової хвилі.

В результаті все Skylake-X стали уніфіковані по дизайну, але зворотною стороною такої уніфікації стало повсюдне застосування дуже великої напівпровідникового кристала з площею близько 485 мм 2, що більш ніж в два з половиною рази перевершує площу кристала восьмиядерних Coffee Lake Refresh. Це означає, що будь-який з LGA2066-процесорів дев'ятитисячний серії має істотно більш високу собівартість в порівнянні з тим же Core i9-9900K. Але незважаючи на це, восьміядерний Core i9-9800X в офіційному прайс-листі оцінений всього на $ 100 дорожче, ніж Core i9-9900K. Тому резонно припустити, що виробництво восьми- і десятіядерних процесорів на основі 18-ядерних кристалів все-таки має для Intel якийсь економічний сенс, наприклад компанія користується цією можливістю для реалізації напівпровідникових кристалів з великим числом виробничих дефектів, які до сих пір не могли знайти гідного застосування.[7]

1.1.2. Процесори десятого покоління

Трохи більше року назад ми познайомилися з процесором Ryzen 9 3950X, який викликав бурю емоцій не тільки у прихильників або супротивників продукції AMD, а й у тих користувачів, які просто зацікавлені у високій продуктивності незалежно від її ідеологічної забарвленості. І це закономірно: флагманська модель для платформи Socket AM4 викликає непідробний інтерес не тільки своїми позамежними для світу масових систем характеристиками, але і тим, як ці характеристики співвідносяться з ціною. Адже якщо після надходження в продаж Ryzen 9 3950X буде продаватися за тією ціною, яку для нього пообіцяла AMD, то вийде, що високопродуктивну робочу станцію рівня HEDT можна буде зібрати в два з гаком рази дешевше, ніж раніше.

Цілком природно, що такі рішучі дії AMD не можуть залишитися без наслідків, тим більше що компанії вдалося вирішити відразу три важливі завдання. По-перше, вона зробила багатоядерні процесори доступними для звичайних користувачів, які зможуть тепер встановлювати їх в звичні масові платформи. По-друге, їй вдалося домогтися того, щоб при виборі між 8 і 12 або 16 ядрами покупці могли керуватися простими міркуваннями доцільності і не відмітали багатоядерні варіанти через їх загороджувальної ціни. І по-третє, заодно AMD запропонувала повністю переосмислити всю концепцію HEDT, зробивши так, щоб ентузіасти, які потребують

позамежних обчислювальних ресурсах і готові вкладати в них значні кошти, змогли тепер розраховувати і на 32-ядерні, і навіть на 64-ядерні процесори.[8][9]

Такі кардинальні зміни кон'юнктури не могли пройти осторонь від Intel, а її платформа LGA2066 виявилася в числі головних «постраждалих» від агресивних маневрів конкурента. Фактично поява доступного 16-ядерник Ryzen 9 3950X розвіяло навколо LGA2066 всю атмосферу преміальності і елітарності, яку мікропроцесорний гігант старанно створював протягом останніх років. Вийшло так, що HEDT-процесори Intel Core X, по суті, перестали бути такими, оскільки новий Ryzen 9 3950X абсолютно на рівних змагається з куди більш дорогими LGA2066-флагманами.

Нові LGA2066-процесори Cascade Lake-X відрізняються від попередників не тільки цінами, хоча це, безумовно, їх головний і самий вражаючий козир. Проте в порівнянні з випущеними рік тому чіпами Core × серії 9000 вони також можуть запропонувати збільшені тактові частоти, поліпшену технологію Turbo Boost 3.0, підтримку великих обсягів швидшої пам'яті, зростання кількості ліній PCI Express 3.0, апаратний захист від деяких вразливостей і нові інструкції , відомі під узагальненою назвою DL Boost (Deep Learning - «глибоке навчання»). При цьому, що зовсім нетипово для Intel, Cascade Lake-X зберігають сумісність з вже випущеними LGA2066-материнськими платами, продовжуючи їх життєвий цикл ще як мінімум на один рік.

Процесори HEDT-сегмента, які Intel пропонує ентузіастам, ще в минулому поколінні цілком переїхали на «середній» за розміром кристал HCC, який використовується в тому числі і в процесорах Xeon з числом ядер менш 18. У новому поколінні Core × в цьому відношенні нічого не змінилося : все Cascade Lake-X засновані на одному і тому ж кремнії, який містить 18 ядер і 24,75 Мбайт L3-кеша. Крім того, характерною особливістю конструкції таких процесорів виступає одноранговая Mesh-мережа, що з'єднує ядра замість звичної звичайні користувачі кільцевої шини. Дана структура, на думку Intel, краще масштабується з ростом кількості ядер.[10]

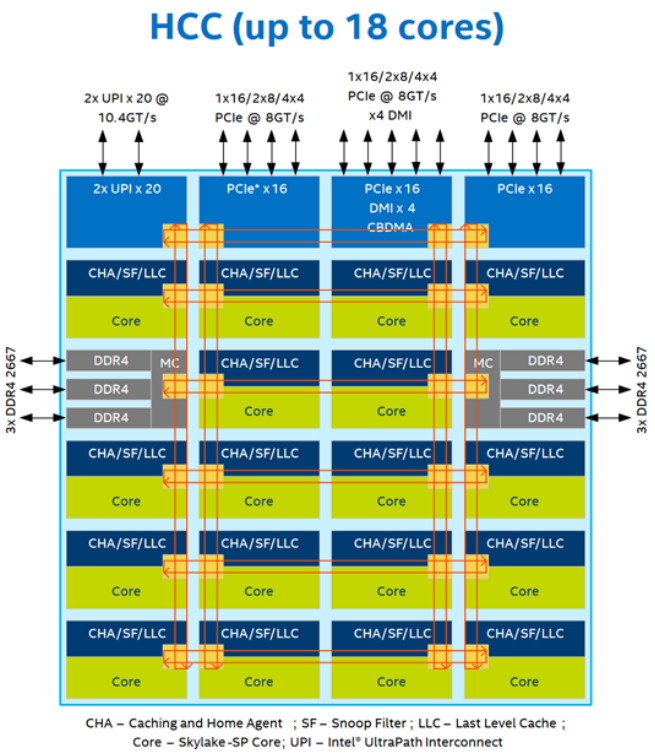


Рисунок 1.6 - архітектура процесорів intel десятого покоління

Також процесори Core-X мають і кілька незвичну підсистему кеш-пам'яті. У них кожне ядро має власний місткий L2-кеш, об'ємом 1 Мбайт (замість 512 Кбайт у Zen 2 і 256 Кбайт у Coffee Lake), але загальний L3-кеш при цьому порівняно невеликою. Його обсяг обчислюється за принципом 1,375 Мбайт на ядро. Втім, все це вж е відомі по минулим поколінням Core-X факти.[11]

1.2. Високопродуктивні процесори AMD

Один з найбільш базових принципів, яким слідує компанія AMD протягом останніх років, звучить так: «А давайте додамо ще ядер». Саме під цим девізом ведуть свій тріумфальний хід як настільні процесори Ryzen разом зі своїми HEDT-побратимами Threadripper, так і серверні чіпи сімейства EPYC. Для наочності можна докладніше згадати про пропозиції AMD для масового сегмента, що з'явилися на ринку протягом останніх років: у першому поколінні Ryzen проти чотирьохядерних Kaby Lake вона виставила процесори з вісьмома ядрами, потім восьмиядерні Ryzen воювали з шестиядерними Coffee Lake, а в цьому році на розправу з восьмиядернікамі Coffee Lake Refresh був кинутий дванадцятиядерний Ryzen 9 3900X. Але на цьому історія про те, наскільки AMD перейнялася ідеєю обсипати користувачів ядрами,

далеко не закінчується, тому що вінцем сімейства Ryzen 3000 компаній вирішила зробити ще більш монструозної процесор - з шістнадцятьма ядрами. незважаючи на збереження тяги до збільшення числа ядер при кожному зручному випадку, теперішня тактика AMD все-таки помітно відрізняється від того, як вона діяла до цього. Якщо в 2017 і 2018 року додаткові ядра в Ryzen виступали якоїсь компенсацією їх більш низькою, ніж у конкурента, питомої продуктивності і частоти, то з архітектурою Zen 2 «червоні» надолужили відставання в показнику IPC і стали претендувати на те, щоб перекроїти під себе весь процесорний ринок. Як з'ясувалося ще чотири місяці тому, коли вийшли 12-ядерні Ryzen 9, за масові процесори AMD з винятковими характеристиками користувачі готові платити значно більше типових для цього ринкового сегмента 500 доларів. Далеко ходити за прикладами не потрібно: коли проблеми з виробництвом Ryzen 9 3900X обернулися суворим дефіцитом і ціна 12-ядерники в піку підскакувала до $ 900, покупців це зовсім не зупиняло - вони продовжували методично змітати їх з прилавків. AMD хоче остаточно закріпити масову платформу Socket AM4 в більш високому позиціонуванні, довівши, що їй не забороняється коштувати в півтора рази більше рекомендованої ціни. Доводити це буде новий флагман Ryzen 9 3950X, в якому число обчислювальних ядер збільшилася ще на щабель - до 16. Такий процесор оцінюється виробником в $ 749, дозволяючи платформі Socket AM4 зробити безсоромну вилазку на територію HEDT, і, судячи з того, що вже було відомо про Ryzen 9 3950X до сьогоднішнього дня, не тільки провести розвідку боєм, але і надійно там влаштуватися. Впевненість в спроможності новинки дають її характеристики. За допомогою сучасного технологічного процесу TSMC з 7-нм нормами і завдяки новаторському чіплетному дизайну компанія AMD змогла зробити так, що збільшення кількості ядер в Ryzen 9 3950X не призводить ні до зниження робочих частот, ні до погіршення енергоефективності. В результаті 16-ядерний флагман повинен гідно проявляти себе у всьому спектрі існуючих завдань, а в ресурсоємких багатопоточних навантаженнях від нього можна очікувати по-справжньому проривний для масового сегмента продуктивності.[12][13]

З точки зору топології новий 16-ядерний Ryzen 9 3950X дуже схожий на розглянутий нами раніше Ryzen 9 3900X. Перед нами - другий масовий процесор

AMD, який грунтується не на двох, а відразу на трьох напівпровідникових кристалах, які називаються в сучасній термінології чіплетами. Один з цих кристалів - універсальний 12-нм чіплет cIOD, що відповідає за функції введення-виведення і містить в собі контроллер пам'яті, контролер PCI Express і елементи SoC. Два інших - 7-нм чіплети CCD, в яких знаходяться обчислювальні ядра, по вісім штук в кожному. Все це поєднується в єдине ціле за допомогою шини Infinity Fabric, яка пов'язує кожен з CCD-чіплетів с кристалом cIOD. При цьому між собою CCD взаємної зв'язку не мають, але це не тягне за собою ніяких негативних наслідків, так як вся логіка Infinity Fabric знаходиться в чіплеті введення-виведення, що зрівнює всі ядра в правах. Іншими словами, Ryzen 9 3950X, на відміну від багатоядерних процесорів Threadripper минулого покоління, не має ніяких NUMA-вузлів і з логічної точки зору має абсолютно монолітним дизайном, в якому затримки при роботі з пам'яттю і при взаємному обміні даними однакові для всіх ядер.[14]

Детальніше про те, як AMD реалізує багаточіплетний підхід при створенні флагманських процесорів Ryzen, ми говорили в огляді 12-ядерного Ryzen 9 3900X. Шіснадцятиядерний Ryzen 9 3950X влаштований саме так, але в ньому використовуються повністю функціональні кристали CCD, в яких немає заблокованих ядер.[15][16]

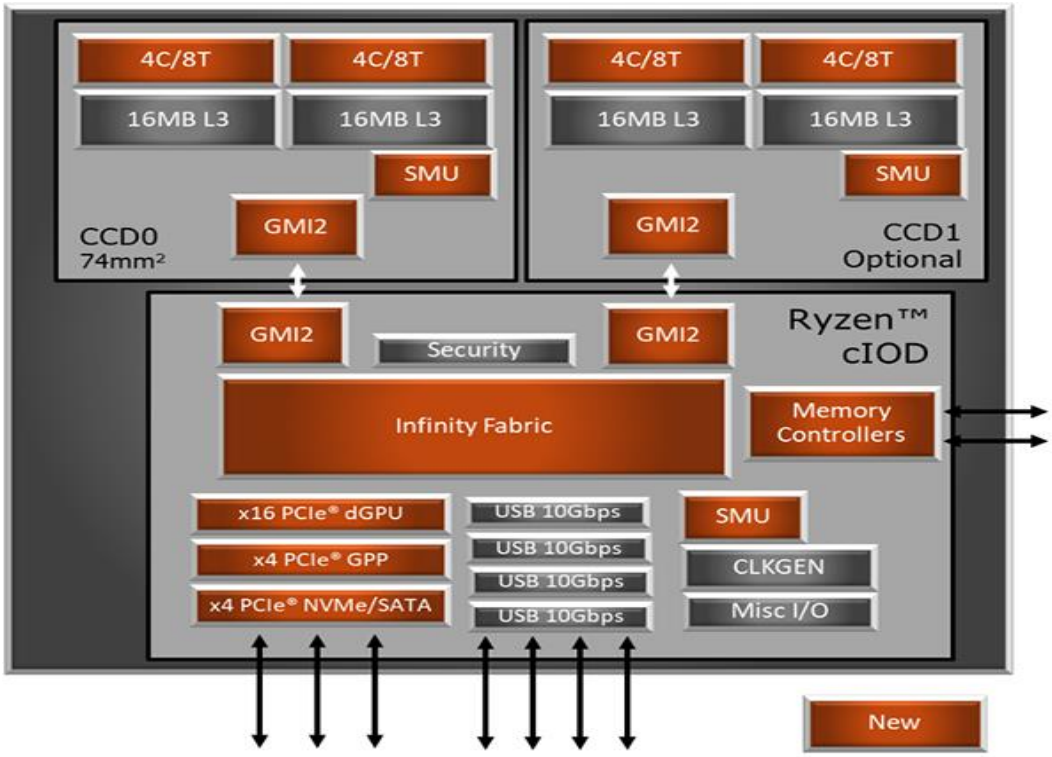


Рисунок 1.7 - архітектура процесорів AMD

Здавалося б, все дуже просто, і зовсім незрозуміло лише те, чому AMD не представила Ryzen 9 3950X раніше, одночасно з іншими Socket AM4-процесорами. Однак у цій затримки є цілком логічне пояснення. Справа в тому, що, отримавши на руки прогресивну мікроархітектуру Zen 2, AMD поставила перед собою амбітну мету не просто принести в масовий сегмент процесор з 16 обчислювальними ядрами, а зробити це якомога ефектніше. Для цього, за початковим задумом, Ryzen 9 3950X повинен був отримати не тільки підтримку максимального для настільного сегмента кількості потоків, а й високі тактові частоти, і щоб все це заодно не приводило до помітного зростання тепловиділення і енергоспоживання.[17]

Багаточіплетна конструкція дозволяє вирішувати різноманітні проблеми, властиві великим монолітним кристалів, завдяки тому, що невеликі за площею чіпи випускати і простіше, і дешевше. Але прямих методів для нейтралізації зростання енергоспоживання і тепловиділення при додаванні в процесор додаткових ядер вона не пропонує. Тому в кінцевому підсумку AMD знадобилося ще кілька часу для того, щоб Ryzen 9 3950X зміг придбати бажані характеристики: 16 ядер, максимальні в усьому модельному ряду Ryzen третього покоління турбочастоти і ординарний для

масових CPU тепловий пакет. Досягається це дуже простим в описі, але досить клопітно в реалізації методом - вибором для таких процесорів найбільш якісних напівпровідникових кристалів.[18]

Насправді такий же підхід вже застосовувався в Ryzen 9 3900X, і на прикладі цього процесора можна було побачити, наскільки непросто дається підбір кристалів CCD для тричіплетних процесорів навіть в тому випадку, якщо потім в них блокується чверть ядер. AMD не вдавалося задовольнити попит і забезпечити безперебійні поставки 12-ядерники протягом цілого кварталу, в результаті чого Ryzen 9 3900X довгий час були в дефіциті. Вибрати ж підходящі кристали для Ryzen 9 3950X ще складніше: два повноцінних восьмиядерних повнофункціональних чіплета CCD разом з кристалом cIOD повинні вписатися в 105-ватний тепловий пакет, забезпечуючи при цьому приблизно такі ж частоти в околиці 4,0 ГГц при повному завантаженні, як і у інших процесорів сімейства Ryzen 3000.

Процесори Threadripper третього покоління отримали в навантаження новий набір системної логіки TRX40, який прийшов на зміну минулого чіпсету X399. Хоча в цілому AMD дотримується політики наступності платформ, в даному випадку їй довелося зробити виняток. Нові Threadripper 3970X і 3960X с платами для минулих HEDT-процесорів компанії несумісні. Одна причина цього полягає в зростанні енергетичних апетитів нових процесорів. Друга - в повному перекладі платформи на підтримку PCI Express 4.0, що зажадало наростити пропускну здатність з'єднання між процесором і чіпсетом в чотири рази (с 3,94 до 15,75 Гбайт / с) з тим, щоб пристрої, за роботу яких відповідає чіпсет, отримували достатню смугу пропускання.[19][20]

1.3 Порівняння процесорів

Для того, щоб зрозуміти котрий процесор найкраще підходить для нашого ПАК. Розглянемо результати тестів даних процесорів на таких додатках:

- 7-Zip Benchmark – тестування на стиснення певної кількості файлів.

- wPrime – тест багатопотоковості процесора.
- X264 pass 1 and 2 – тест на швидкодію системи.
- Cinebench r10 – тест на апаратні можливості процесора
- Cinebench r15 single - тест на апаратні можливості процесора при одному потоці
- Cinebench r15 multi - тест на апаратні можливості процесора при багатопотоковості
- Cinebench r20 multi - тест на апаратні можливості процесора при багатопотоковості

Таблиця 1.1 – Порівняння процесорів за допомогою тестів

Процесор	7-Zip	wPRIME	X264 Pass 1	X264 Pass 2	Cinebench R10	Cinebench R15 single	Cinebench R15 multi	Cinebench R20 multi
AMD R9 3950X	99467	4.11	265	168	53396	209	4002	9075
AMD R9 3900	81119	4.19	205	267	45058	209	3049	6910
AMD 1950	93099	2.4	224	167	61666	198	3907	8857
AMD 2950	82119	2.56	267	147	45075	209	3049	6920
I9-10980XE	93076	2.71	215	157	59476	208	3744	8532

процесори	z-zip	wPRIME	X264 Pass 1	X264 Pass 2	Cinebench R10	Cinebench R15 single	Cinebench R15 multi	Cinebench R20 multi
I9-9980XE	93099	2.41	224	167	61666	198	3904	8857
I7-6950X	6586	3.5	196	101	40378	155	1859	7021

Висновок до розділу 1

В ході виконання даного розділу, було розглянуто різні багатоядерні процесори , які чудово підходять для паралельного програмування.

У ході виконання програми бувають випадки, коли праграмі потрібно очистити пам’ять і добавити нову інформацію. В цей час більшість ядер процесора не можуть швидко виконувати свою роботу, і швидкість роботи програми замітно знижується.

Threadripper 1950x можна назвати кращим багатоядерним процесом із розглянутих нами. За свою невелику ціну в 400 доларів, ми отримуємо потужний процесор, який показує хороші результати як в багато поточних задачах, так і не відстає від дорожчих аналогів в потужності на 1 ядро, а великий об’єм кеш пам’яті дозволяє нам уникнути проблем із її заповненням.

РОЗДІЛ 2. ПРОГРАМНІ ЗАСОБИ ПОБУДОВИ ВИСОКОПРОДУКТИВНИХ КОМПЛЕКСІВ

2.1. Мови паралельного програмування

Паралельне програмування останнім часом стає все більш необхідне, завдяки темпам розвитку багатоядерних процесорів. Одним з варіантів організації паралельного програмування являється багатопотокове програмування. В звичайній програмі виконується тільки один потік управління а в багатопоточній одночасно може працювати декілька потоків.

В багатопоточних програмах ускладнюється контроль за обміном даних між потоками. Глобальні ресурси необхідно захищати від одночасного доступу зі сторони декількох потоків, щоб не пошкодити їх цілісність.

В даному розділі розгляну мови паралельного програмування такі як: Java, C#, Ada, Python, та бібліотеки програмування:OpenMP, MPI, WinAPI.

2.1.1. Java

В мові програмування Java можливо створити потоки двома способами:

1) За допомогою інтерфейсу Runnable.
2) Наслідуючи клас Thread.

Клас Thread являє собою набір методів для керування потоками (табл 2.1)

Таблиця 2.1 – методи класу Thread

Методи класу Thread	Операції
getName()	Отримання імені потоку
getPriority()	Отримання пріоритету потоку
isAlive()	Перевірка виконання потоку
join()	Очікування завершення потоку
run()	Точка входу в потік
sleep()	Призупинення потоку на заданий час
start()	Запуск потоку

Розберем більш детально методи класу такі як getPriority() та sleep().За допомогою getPriority() можна задати пріоритети потоків від 1 до 10. При цьому за умовчуванням потік отримує значення 5 – нормальний пріоритет. Використовуючи метод sleep() можна призупинити потік на зазначений проміжок часу. Параметр час затримки указується в мілісекундах.

Є більш складний варіант створення потоків. Для створення якого потрібно реалізувати інтерфейс Runnable. Можна створити потік з любого об'єкту, який реалізує інтерфейс Runnable і об'явити метод run().

Thread (Runnable об'єкт_потоку, String назва_потоку)

В першому параметрі вказуємо екземпляр класу, що реалізує інтерфейс. Він визначає де починається виконання потоку. В другому параметрі передається ім'я потоку.

Всередині конструктора Runnable() потрібно створити об'єкт класу Thread.

Thread = new Thread(this, " потік для прикладу");

В першому параметрі використовується об'єкт this. Що означає мету викликати метод run() цього об'єкту. Далі викликається метод start(), в результаті чого запускається виконання потоку, починаючи з методу run(). В свою чергу метод запускає цикл для потоку. Після виклику методу start(), конструктор Runnable() повертає управління додатку. Коли головний потік продовжує виконувати свою роботу, він входить в свій цикл. Після чого обидва потоки виконуються паралельно.

Можливо запускати декілька потоків одночасно, а не тільки другий потік в доповнення до першого, проте це може призвести до конфліктів, коли два потоки пробують працювати з одною змінною одночасно.

Для рішення цієї проблеми було добавлено метод synchronized. Тобто метод може мати модифікатор synchronized . Коли потік знаходиться в середині синхронізованого методу, всі інші потоки, котрі намагаються визвати його в тому ж

екземплярі , повинні очікувати. Це дозволяє виключити мішанину, коли декілька потоків намагаються доступитись до одного метода.[21]

2.1.2 C#

Основний функціонал для використання потоків в мові програмування C# зосереджений в System.Threading. В ньому визначений клас, який являє собою окремий потік – клас Thread.

Клас Thread визначає ряд методів і властивостей, які дозволяють управляти потоками та отримувати інформацію про них.

Основні властивості класу:

- Статична властивість CurrentContext дозволяє отримат контекст, в якому виконується потік
- Статична властивість CurrentThread повертає посилання на виконуваний потік
- Властивість IsAlive указує, чи виконується потік в данний момент
- Властивість IsBackground вказує, чи являється потік фоновим
- Властивість Name включає в себе ім’я потоку
- Властивість Priority зберігає пріоритет потоку – значення перечислення ThreadPriority
- Властивість ThreadState повертає стан потоку – одне з значень перечислення ThreadState

Методи класу Thread:

- Статичний метод GetDomain повертає ссилку на домен програми
- Статичний метод GetDomainId повертає id домену програми, в якій виконується даний потік
- Статичний метод Sleep зупиняє потік на визначену кількість мілісекунд
- Метод Abort повідомляє осередок CLR про те що потрібно зупинити потік, проте призупинення роботи виконується не відразу, а тільки тоді коли це стає можливим.

- Метод Interrupt прирівнює потік на деякий час
- Метод Join блокує виконання викликаного його потоку до того часу, поки не завершиться потік, для якого був викликаний даний метод
- Метод Start запускає потік на виконання

Статуси потоку знаходяться в переліку ThreadState:

- Aborted: поток призупинений, але поки що не завершений до кінця
- AbortRequested: для потоку викликаний метод Abort, але завершення потоку ще не відбулась
- Stopped: потік завершився
- StopRequested: потік отримав запит на завершення
- Suspended: потік призупинено
- SuspendRequested: потік отримав запит на призупинення
- Unstarted: потік не був запущений
- WaitSleepJoin: потік заблокований в результаті дії методів Sleep або Join

Пріорітети потоків знаходяться в переліку ThreadPriority:

- Lowest
- BelowNormal
- Normal
- AboveNormal
- Highest

По замовчуванню потоку задається значення Normal.

Для роботи з потоками, які доступаються до одного і того ж ресурсу використовується ключове слово lock. Оператор lock визначає блок коду, всередині якого весь код блокується і стає недоступним для інших потоків до завершення даного потоку.

Також для синхронізації потоків можна також використовувати монітори, представлені класом System.Threading.Monitor. Фактично конструкція lock з попередньої теми інкапсулює в себе синтаксис виконання моніторів.

```
Monitor.Enter(locker, ref acquiredLock);
```

Метод Monitor.Enter приймає два параметри – об’єкт блокування і значення типу bool, котре вказує на результат блокування(якщо воно true, то блокування виконано успішно). Фактично метод блокує об’єкт locker так само, як це робить оператор lock. За допомогою Monitor.Exit виконується звільнення об’єкту locker, якщо розблокування успішне, то він стає видимим для інших потоків.

Крім блокування і розблокування об’єкта класу Monitor має ще ряд методів, яку дозволяють управляти синхронізацією потоків. Так, метод Monitor.Wait звільняє блокування об’єкту і переводить потік в чергу очікування об’єкта. Наступний потік в черзі готовності об’єкту блокує даний об'єкт. А всі потоки. які визвали метод Wait, залишаються в черзі очікування, поки не отримають сигнал від метода Monitor.Pulse або Monitor.PulseAll, відправленого власником блокування. Якщо метод Monitor.Pulse відправив сигнал, то потік, який знаходився перший в черзі очікування, отримує сигнал і блокує метод котрий звільнився. Якщо ж метод Monitor.PulseAll відправлений, то всі потоки, котрі знаходяться в черзі очікування, отримують сигнал і переходять в чергу готовності, де їм знову дозволяється отримувати блокування об’єкту.[22]

2.1.3 Ada

Ада – мова програмування призначена для розроблення програмних систем з високою надійністю. Процеси в ній реалізуються як задачі(Task).Task – один з видів модулів мови, описується у вигляді специфікації і тіла. Задачний тип надає змогу користувачу описати тип об’єктами якого будуть задачі.

Пріоритети в Ada задаються за допомогою прагми Priority. Пріоритет задачі знаходиться в проміжку від 1 до 7. Пріоритет визначає спроможність задачі в боротьбі за ресурси. Призупинення задачі виконується за рахунок оператора delay, в якому час очікування вказується в секундах.

Концепція рандеву

Як один з механізмів забезпечення надійного міжзадачного обміну даними і обоюдної синхронізації роботи задач. Ада надає механізм рандеву. Основна ідея механізму рандеву досить проста. В специфікації задачі вказуються різні точки входу (entry) в задачу, в яких вона готова очікувати звернення до неї від інших задач. Далі в тілі задачі вказуються інструкції прийняття звернень до відповідних входів, вказаним в специфікації цієї задачі.

Необхідно звернути увагу на несиметричність такого механізму взаємодії. Це означає, що в процесі взаємодії одної з задач розглянутої як сервер, а іншої – як клієнта, причому задача-сервер не може бути ініціатором початку взаємодії.

В найпростішому випадку, коли розглядається взаємодія тільки двох задач, задача-клієнт, яка бажає звернутись до іншої задачі(задачі-серверу), ініціалізує звернення до входу задачі-сервера. Після чого задача відгукується на виклик задачі-клієнта, приймаючи звернення до даного входу. Таким чином, взаємодія двох задач здійснюється в ситуації, коли задача-клієнт звернулась до входу, а задача-сервер готова прийняти це звернення. Цей спосіб взаємодії двох задач називається рандеву.

Оскільки завдання-клієнт і завдання-сервер виконуються незалежно один від одного, то немає ніякої гарантії, що обидві задачі виявляться в точці здійснення рандеву одночасно. Тому, якщо завдання-сервер виявилася в точці рандеву, але при цьому немає жодного звернення до входу (запиту на взаємодію), то вона повинна чекати появи такого звернення. Аналогічно, якщо завдання-клієнт звертається до входу, а завдання-сервер не готова обслужити таке звернення, то завдання-клієнт повинна чекати, поки завдання-сервер обслужить це звернення. В процесі очікування як завдання-клієнт, так і завдання-сервер не займають ресурси процесора, перебуваючи в стані, який називають припиненим або станом блокування.

Для опису входів завдання-сервера використовується зарезервоване слово entry, а семантика входів завдань дуже схожа на семантику процедур:

- Так само, як і процедури, входи завдань мають імена і можуть мати різні параметри. Для імен входів допускається суміщення імен, що має на увазі

наявність у одного завдання декількох входів з однаковими іменами, але різними параметрами.

- Параметри входів завдання, так само, як і параметри процедур, можуть використовуватися в режимах "in", "in out" і "out", і можуть мати значення за замовчуванням.[23]

2.1.4 Python

Для створення потоків в python використовується модуль threading. Є два варіанта створення потоків:

Виклик фунцкції:

Threading.Thread()

Виклик класу:

Threading.thread

Для керування потоками існують методи:

- Start() – запускає потік
- Run() – метод надає інструкції, які виконуються в потоці.
- Join([timeout]) – потік, який викликає цей метод, призупиняється, та очікує закінчення потоку. Параметр timeout (число з плаваючою комою) дозволяє указати час очікування в секундах, по закінченню якого призупинений потік продовжує роботу незалежно від завершення потоку, чий метод join був викликаний. Викликати join() деякого потоку можна викликати декілька разів. Потік не може визвати метод join() самого себе. Також неможна очікувати завершення ще не запущеного потоку.[24]
- getName() – повертає ім’я потоку.
- setName(name) – присвоює потоку ім’я name.
- isAlive() – повертає істину, якщо потік працює (метод run() уже викликаний)
- isDaemon() – повертає істину, якщо потік має признак daemonic.

- setDaemon(daemonic) – встановлює признак того, що потік являється daemonic
- activeCount() – повертає кількість активних в даний момент екземплярів класу Thread
- currentThread() – повертає поточний об’єкт-потік, тобто відповідний потоку управління, який викликав цю функцію
- enumerate() – повертає список активних потоків.

Рішення проблеми доступу до спільних ресурсів за допомогою threading.lock.

Об’єкт Lock має методи:

- acquire([blocking=True]) – робить запит на запирання замка, якщо параметр blocking не вказаний або являється істинною, то потік буде очікувати звільнення замка. Якщо blocking був заданний і був істинною, то метод поверне True. Якщо блокування непотрібне, то метод верне True, якщо замоку був не зачинений і ним успішно управляє даний потік. В інакшому випадку буде повернено False.
- release() – запрос на відпирання замка.
- locked() – повертає теперішній стан замка.

2.2 Бібліотеки паралельного програмування

У данному підрозділі розгялну найбільш популярні бібліотеки для паралельного програмування такі, як: MPI, WinAPI, OpenMP.

2.2.1 MPI

MPI розшифровується, як Message Passing Interface – це бібліотека, що забезпечує обмін повідомленнями між процесами.[25]

Повідомлення MPI мають такі компоненти:

- блок даних
- тип даних
- кількість даних

Наявна така інформація про відправника і одержувача:

- комунікатор групи працюючих паралельних процесів
- ранг відправника
- ранг одержувача

Обмін повідомлення має чотири форми:

- попарний – два процеси передають між собою
- колективний – один процес відправляє всім, що знаходяться в одній групі об’єднаній комунікатором.
- Асинхронний – процес, що відправляє не чекає отримання процесом-одержувачем повідомлення. (в цьому випадку повідомлення копіюється в буфер)
- Синхронний – процес-відправник блокується поки процес-одержувач не отримає повідомлення, і навпаки.

Функції бібліотеки MPI:

- MPI_Send() – відправити повідомлення. (блокування поки повідомлення не буде отримано)
- MPI_Recv()- Отримати повідомлення (блокувати поки не відправиться повідомлення)
- MPI_ISend() – пересилання повідомлення без блокування
- MPI_IRecv() – отримання повідомлення без блокування
- MPI_Probe() – дізнатись чи повідомлення було отримано
- MPI_IProbe() - дізнатись чи повідомлення було отримано (без блокування)
- MPI_Bcast() – надсилання повідомлення всім процесам в групі
- MPI_Reduce() – отримати дані від всіх у групі
- MPI_Gather() – зібрати дані з всіх процесів
- MPI_Scatterv() – розіслати дані всім процесам у групі (дані одного розміру)

- MPI_Scatterv() – розіслати дані всім процесам у групі (дані різного розміру)
- MPI_Alltoall() - розіслати дані всім процесам в групі зі всіх процесів групи. (дані одного розміру)
- MPI_Alltoallv() - розіслати дані всім процесам в групі зі всіх процесів групи. (дані різного розміру)

2.2.2. WinAPI

Потік по суті є шляхом виконання програми. Це також найменша одиниця виконання, яку планує Win32. Потік складається з стека, стану регістрів ЦП і записи в списку виконання планувальника системи. Кожен потік використовує всі ресурси процесу.

Процес складається з одного або декількох потоків, а також коду, даних і інших ресурсів програми в пам'яті. Типовими ресурсами програми є відкриті файли, семафори і динамічно виділяється пам'ять. Програма виконується, коли планувальник системи надає один зі своїх потоків управління виконанням. Планувальник визначає, які потоки повинні виконуватися і коли вони повинні виконуватися. Потоки з більш низьким пріоритетом можуть очікувати, поки потоки з більш високим пріоритетом завершать свої завдання. На багатопроцесорних комп'ютерах планувальник може переміщати окремі потоки на різні процесори для балансування навантаження ЦП.

Кожен потік процесу працює незалежно один від одного. Якщо ви не зробите їх видимими, потоки виконуються окремо і не знають про інші потоки в процесі. Однак потоки, спільно використовують загальні ресурси, повинні координувати свою роботу з допомогою семафорів або іншого методу взаємодії між процесами.

Кожен потік має власний стек і його власну копію регістрів ЦП. Інші ресурси, такі як файли, статичні дані і пам'ять купи, спільно використовуються всіма потоками в процесі. Потоки, що використовують ці загальні ресурси, повинні бути синхронізовані. Win32 надає кілька способів синхронізації ресурсів, включаючи семафори, критичні розділи, події і м'ютекси.

Якщо кілька потоків звертаються до статичних даних, програма повинна надати доступ до можливих конфліктів ресурсів. Розглянемо програму, в якій один потік оновлює статичну структуру даних, що містить координати x, y для елементів, що відображаються іншим потоком. Якщо потік поновлення змінює координату x і завантажується до того, як він може змінити координату y, то потік виведення може бути запланований до поновлення координати y. Елемент відображається в неправильному місці. Цю проблему можна уникнути, використовуючи семафори для управління доступом до структури.

Мютекс - це спосіб обміну даними між потоками або процесами, які виконують асинхронне взаємодія один з одним. Ця взаємодія можна використовувати для координації дій кількох потоків або процесів, зазвичай шляхом управління доступом до загального ресурсу шляхом блокування і розблокування ресурсу. Щоб вирішити цю проблему відновлення координат x, y, потік поновлення встановлює м'ютекс, який вказує, що структура даних використовується до виконання оновлення. Він очистить мютекс після обробки обох координат. Перш ніж оновити відображення, потік виведення повинен дочекатися очищення мьютекса. Цей процес очікування мютекса часто називається блокуванням мьютекса, оскільки процес блокується і не може тривати до тих пір, поки м'ютекс не очистите.[26]

Розлянем функції доступні в WinAPI для роботи з потоками даних. Для того щоб створити потік використовується фунції CreateThread.

Функції CreateThread:

- StackSize – початковий розмір стеку (вказується в байтах). Якщо в параметрах вказати розмір стеку 0, то система задасть розмір стеку за замовчуванням.
- IpStartFuncAddr – це вказівник на функцію, яка буде виконуватись потоком.
- dwCreatParam – параметри, які управлють створенням потоку.
- Thrld – вказівник на змінну, в котру буде записаний ідентифікатор потоку.
- ExitThread – функція закриваюча потік

- dwExitCode – код закінчення потоку
- SuspendThread(HANDLE thread) – дескриптор потоку, який необхідно призупинити. Він повинен мати права доступу: THREAD_SUSPEND_RESUME.
- ResumeThread(HANDLE thread) - дескриптор потоку, який необхідно призупинити. Він повинен мати права доступу: THREAD_SUSPEND_RESUME

2.2.3 OpenMP

OpenMP - механізм написання паралельних програм для систем зі спільною пам'яттю.Складається з набору директив компілятора і бібліотечних функцій.

Конструкції OpenMP

Умови виконання визначають то, як буде виконуватися паралельний ділянку коду і область видимості змінних всередині цієї ділянки коду. Наведемо наступні умови:

shared (змінна 1, змінна 2, ...)

Умова shared вказує на те, що всі перераховані змінні будуть розділятися між потоками. Всі потоки будуть доступать до однієї і тієї ж області пам'яті.

private (змінна 1, змінна 2, ...)

Умова private вказує на те, що кожен потік повинен мати свою копію змінної на всьому протязі свого виконання.

Firstprivate (змінна 1, змінна 2, ...)

Ця умова аналогічно умові private за тим винятком, що зазначені змінні присвоюються при вході в паралельний ділянку коду значенням, яке мала змінна до входу в паралельну секцію.

lastprivate (змінна 1, змінна 2, ...)

Приватний змінні зберігають своє значення, яке вони отримали при досягненні кінця паралельної ділянки коду.

reduction (оператор: змінна 1, змінна 2, ...)

Ця умова гарантує безпечне виконання операцій редукції, наприклад, обчислення глобальної суми.

if (вираз)

Ця умова говорить про те, що паралельне виконання необхідно тільки якщо вираз істинний.

default (shared | private | none)

Ця умова визначає зону видимості змінних всередині паралельної ділянки коду за замовчуванням.

schedule (type [, chank])

Цим умовою контролюється те, як ітерації циклу розподіляються між потоками

Засоби синхронізації:

У OpenMP передбачені наступні конструкції синхронізації:

critical - критична секція

barrier - точка синхронізації

master - блок, який буде виконаний тільки основним потоком. Всі інші потоки пропустять цей блок. В кінці блоку неявної синхронізації немає.

ordered - виконувати блок в заданій послідовності

flush - негайне скидання значень роздільних змінних в пам'ять.

Наявність критичної секції в паралельному блоці гарантує, що вона в кожен конкретний момент часу буде виконуватися тільки одним потоком. Тобто коли один потік знаходиться в критичній секції, всі інші потоки, які готові в неї увійти, знаходяться в загальмованому стан. Критичні секції можуть забезпечуватися

іменами. При цьому критичні секції вважаються незалежними, тільки якщо вони використовують різні імена. За замовчуванням, всі непроіменовані критичні секції мають одне ім'я.[27]

Висновок до розділу 2

В ході виконання даного розділу моєї дипломної роботи, було розглянуто декілька прикладів найпопулярніших мов та бібліотек, які використовуються для створення паралельних програм.

Паралельні програми працюють на усіх платформах, з будь-якою операційною системою, якщо на даній платформі встановлена віртуальна машина Java.

У будь-якої мови високого рівня досить низька продуктивність через компіляції та абстракції за допомогою віртуальної машини. Однак це не єдина причина низької швидкості Java. Наприклад, додаток очищення пам'яті: це корисна функція, яка, на жаль, призводить до значних проблем з продуктивністю, якщо вимагає більше 20 відсотків часу процесора. Погана настройка кешування може викликати надмірне використання пам'яті. Існує також взаємне блокування потоків: так відбувається, коли кілька потоків намагаються отримати доступ до одного і того ж ресурсу. В цьому випадку відбувається кошмар кожного Java-розробника - помилка через брак пам'яті. Проте вміле планування може вирішити всі ці проблеми.

РОЗДІЛ 3.СТВОРЕННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ ДЛЯ ВИРІШЕННЯ ЗАДАЧ ЛІНІЙНОЇ АЛГЕБРИ НА ОСНОВІ ІСНУЮЧИХ РІШЕНЬ

3.1 Створення структури комплексу

Основним критерієм, при підборі комплектуючих для нашого комплекса була максимальна потужність за свою вартість. Threadripper 1950x став кращим процесором для наших завдань. Материнська плата ASUS Zenith Extreme (X399) оснащена сучасними слотами розширення. 4 модуля оперативної пам’яті дозволять нашій системі працювати з максимальною пропускною потужністю. Be Quiet! Pure Power 11 600W тихий і потужний блок живленя, з усіма потрібними схемами захисту. Samsung 850 EVO швидкісний ссд, який збільшить швидкість роботи нашої системи.

	Найменування	Ціна(грн)
Процесор	Threadripper 1950x	10 800
Материнська плата	ASUS Zenith Extreme (X399)	15 850
Оперативна пам’ять	AMD Radeon DDR4 4x4GB 2400Mhz	3220
Блок живлення	Be Quiet! Pure Power 11 600W	2500
Відеокарта	NVIDIA GeForce 1080 GTX TI	12500
Жорсткий диск	Samsung 850 EVO 500 GB	2500
Корпус	Deepcool MATREXX 55	1322

3.2 Розробка програмного забезпечення комплексу

У даному розділі розроблено програму для паралельної комп`ютерної системи зі спільною пам`яттю, яка обраховує заданий математичні вирази:

1)MA = sort(MO)*ME+min(Z)*MR*e

$b_i = \min(Z_H)$ $i = \underline{1, P}$

$b = \min(b, b_i)$

$MO_H^C = sort^*(MO_H)$

$MA_H = MO^C * ME_H + b * MR_H * e$

де:

H = N/P

i=1..P;

Спільний ресурс: e, MO^C, b.

Sort*- сортування

2)MA = max(Z)*(MO*MK) – d*MS

$l_i = \max(Z_H)$

$l = \max(l, l_i)$

$MA_H = l*(MO_H*MK) – d*MS_H$

Спільні ресурси: l, MK, d

Пояснення до використовуваних констант:

N – розмірність векторів і матриць;

P – кількість потоків;

$H = \frac{N}{P}$.

3.2.2. Розробка алгоритмів задач

Дана програма реалізується чотирма потоками й алгоритм виконання процесів показано в таблиці 2.1.

Task1	Точки синх.	Task2	Точки синх.	Task3	Точки синх.	Task4	Точки синх.
						1. Введення MO,ME,Z,MR ,e	
						2. Сигнал T1, T2, T3 о введення	$S_{4,1}$ $S_{5,2}$ $S_{5,3}$
1. Чекати сигнал від T4 про завершення введення	$W_{1,1}$	1. Чекати сигнал від T4 про завершення введення	$W_{2,1}$	1. Чекати сигнал від T4 про завершення введення	$W_{3,1}$		
2. $b_1 = \min(Z_H)$		2. $b_2 = \min(Z_H)$		2. $b_3 = \min(Z_H)$		3. $b_4 = \min(Z_H)$	
3. $b = \min(b,b_1)$	КУ	3. $b = \min(b,b_2)$	КУ	3. $b = \min(b,b_3)$	КУ	4. $b = \min(b,b_4)$	КУ
4. Сигнал T2, T3, T4 про завершення обчислень b	$S_{1,1}$ $S_{1,2}$ $S_{1,3}$	4. Сигнал T1, T3, T4 про завершення обчислень b	$S_{2,1}$ $S_{2,2}$ $S_{2,3}$	4. Сигнал T1, T2, T4 про завершення обчислень b	$S_{3,1}$ $S_{3,2}$ $S_{3,3}$	5. Сигнал T1, T2, T3 про завершення обчислень b	$S_{4,4}$ $S_{4,5}$ $S_{4,6}$

Task1	Точки синх.	Task2	Точки синх.	Task3	Точки синх.	Task4	Точки синх.
5. Чекати сигнал від T2, T3, T4 про завершення обчислень b	$W_{1,2}$ $W_{1,3}$ $W_{1,4}$	5. Чекати сигнал від T1, T3, T4 про завершення обчислень b	$W_{2,2}$ $W_{2,3}$ $W_{2,4}$	5. Чекати сигнал від T1, T2, T4 про завершення введення b	$W_{3,2}$ $W_{3,3}$ $W_{3,4}$	6. Чекати сигнал від T1, T2, T3 про завершення введення b	$W_{4,1}$ $W_{4,2}$ $W_{4,3}$
6. MO_H^C = sort*(MO_H)		6. MO_H^C = sort*(MO_H)		6. MO_H^C = sort*(MO_H)		7. MO_H^C = sort*(MO_H)	
7. Сигнал T2, T3, T4 про завершення сортування MO_H^C	$S_{1,4}$ $S_{1,5}$ $S_{1,6}$	7. Сигнал T1, T3, T4 про завершення сортування MO_H^C	$S_{2,4}$ $S_{2,5}$ $S_{2,6}$	7. Сигнал T1, T2, T4 про завершення сортування MO_H^C	$S_{3,4}$ $S_{3,5}$ $S_{3,6}$	8. Сигнал T1, T2, T3 про завершення сортування MO_H^C	$S_{4,7}$ $S_{4,8}$ $S_{4,9}$
8. Чекати сигнал від T2, T3, T4 про завершення сортування MO_H^C	$W_{1,5}$ $W_{1,6}$ $W_{1,7}$	8. Чекати сигнал від T1, T3, T4 про завершення сортування MO_H^C	$W_{2,5}$ $W_{2,6}$ $W_{2,7}$	8. Чекати сигнал від T1, T2, T4 про завершення сортування MO_H^C	$W_{3,5}$ $W_{3,6}$ $W_{3,7}$	9. Чекати сигнал від T1, T2, T3 про завершення сортування MO_H^C	$W_{4,4}$ $W_{4,5}$ $W_{4,6}$
9. Копія e1 = e	КУ	9. Копія e2 = e	КУ	9. Копія e3 = e	КУ	10. Копія e4 = e	КУ
10. Копія MO^C1 = MO^C	КУ	10. Копія MO^C2 = MO^C	КУ	10. Копія MO^C3 = MO^C	КУ	11. Копія MO^C4 = MO^C	КУ
11. Копія b1 = b	КУ	11. Копія b2 = b	КУ	11. Копія b3 = b	КУ	12. Копія b4 = b	КУ

Task1	Точки синх.	Task2	Точки синх.	Task3	Точки синх.	Task4	Точки синх.
12. Обчислення MA_H = MO^C1*ME_H + $b1*MR_H*e1$		12. Обчислення MA_H = MO^C2*ME_H + $b2*MR_H*e2$		12. Обчислення MA_H = MO^C3*ME_H + $b3*MR_H*e3$		13. Обчислення MA_H = MO^C4*ME_H + $b*MR_H*e$	
13. Сигнал Т4 про завершення обчисленнь	$S_{1,7}$	13. Сигнал Т4 про завершення обчисленнь	$S_{2,7}$	13. Сигнал Т4 про завершення обчисленнь	$S_{3,7}$	14. Чекати сигнал від Т1 , Т2 , Т3 про завершення обчислень	$W_{4,7}$
						15. Вивід МА	

Таблиця 3.2. Алгоритм роботи потоків (на прикладі чотирьох потоків)

Задача Т1	ТС/КУ
1. Введення Z	
2. Сигнал про завершення вводу в Т2. Т3, Т4, Т5, Т6	$S_{2,3,4,5,6-1}$
3. Чекати завершення вводу в Т3. Т6	$W_{3,6-1}$
4. Обчислення 1 l_1 = max(Z_H)	
5. Обчислення 2 l = max(l, l_1)	КУ
6. Сигнал про завершення про завершення обчислення 1, 2 в Т2. Т3. Т4. Т5. Т6	$S_{2,3,4,5,6-2}$
7. Чекати завершення обчислення 1..2 в Т2. Т3. Т4. Т5. Т6	$W_{2,3,4,5,6-2}$
8. Копіювання l1=l, d1=d, MK1=MK	КУ

Задача Т1	
9. Обчислення 3 MA_H = l1*(MO_H*MK1) - d1* MS_H	
10. Чекати завершення обчислення 3 в Т2. Т3. Т4. Т5. Т6	$W_{2,3,4,5,6-3}$
11. Вивести МА	
Задача Т2	
1. Чекати завершення вводу в Т1. Т3. Т6	$W_{1,3,6-1}$
2. Обчислення 1 l_1 = max(Z_H)	
3. Обчислення 2 l = max(l, l_1)	КУ
6. Сигнал про завершення про завершення обчислення 1, 2 в Т1, Т3. Т4. Т5. Т6	$S_{1,3,4,5,6-1}$
7. Чекати завершення обчислення 1. 2 в Т1. Т3. Т4. Т5. Т6	$W_{1,3,4,5,6-2}$
8. Копіювання l2=l, d2=d, MK2=MK	КУ
9. Обчислення 3 MA_H = l2*(MO_H*MK2) - d2* MS_H	
10. Сигнал про завершення обчислення 3 в Т1	S_{1-2}
Задача Т3	
1. Введення МО, MS	
2. Сигнал про завершення вводу в Т1. Т2, Т4, Т5, Т6	$S_{1,2,4,5,6-1}$
3. Чекати завершення вводу в Т1. Т6	$W_{1,6-1}$
4. Обчислення 1 l_1 = max(Z_H)	
5. Обчислення 2 l = max(l, l_1)	КУ
8. Сигнал про завершення про завершення обчислення 1, 2 в Т1, Т2. Т4. Т5. Т6	$S_{1,2,4,5,6-2}$
9. Чекати завершення обчислення 1. 2 в Т1. Т2. Т4. Т5. Т6	$W_{1,2,4,5,6-2}$
10. Копіювання l3=l, d3=d, MK3=MK	КУ
11. Обчислення 3 MA_H = l3*(MO_H*MK3) - d3* MS_H	
12. Сигнал про завершення обчислення 3 в Т1	S_{4-3}
Задача Т4	
1. Чекати завершення вводу в Т1. Т3. Т6	$W_{1,3,6-1}$
2. Обчислення 1 l_1 = max(Z_H)	

Задача Т4	
3. Обчислення 2 l = max(l, l_1)	КУ
6. Сигнал про завершення про завершення обчислення 1, 2 в Т1, Т2. Т3. Т5. Т6	$S_{1,2,3,5,6-1}$
7. Чекати завершення обчислення 1. 2 в Т1. Т2. Т3. Т5. Т6	$W_{1,2,3,5,6-2}$
8. Копіювання l4=l, d4=d, МК4=МК	КУ
9. Обчислення 3 MA_H = l4*(MO_H*МК4) - d4* MS_H	
10. Сигнал про завершення обчислення 3 в Т1	S_{1-2}
Задача Т5	
1. Чекати завершення вводу в Т1. Т3. Т6	$W_{1,3,6-1}$
2. Обчислення 1 l_1 = max(Z_H)	
3. Обчислення 2 l = max(l, l_1)	КУ
6. Сигнал про завершення про завершення обчислення 1, 2 в Т1, Т2. Т3. Т4. Т6	$S_{1,2,3,4,6-1}$
7. Чекати завершення обчислення 1. 2 в Т1. Т2. Т3. Т4. Т6	$W_{1,2,3,4,6-2}$
8. Копіювання l5=l, d5=d, МК5=МК	КУ
9. Обчислення 3 MA_H = l5*(MO_H*МК5) - d5* MS_H	
10. Сигнал про завершення обчислення 3 в Т1	S_{1-2}
Задача Т6	
1. Введення МК, d	
2. Сигнал про завершення вводу в Т1. Т2, Т3, Т4, Т5	$S_{1,2,3,4,5-1}$
3. Чекати завершення вводу в Т1. Т3	$W_{1,3-1}$
4. Обчислення 1 l_1 = max(Z_H)	
5. Обчислення 2 l = max(l, l_1)	КУ
8. Сигнал про завершення про завершення обчислення 1, 2 в Т1, Т2. Т3. Т4. Т5	$S_{1,2,3,4,5-2}$
9. Чекати завершення обчислення 1. 2 в Т1. Т2. Т3. Т4. Т5	$W_{1,2,3,4,5-2}$
10. Копіювання l6=l, d6=d, МК6=МК	КУ
11. Обчислення 3 MA_H = l6*(MO_H*МК6) - d6* MS_H	

Задача Т6	
12. Сигнал про завершення обчислення З в Т1	S_{4-3}

3.2.3 Розробка схеми взаємодії задач

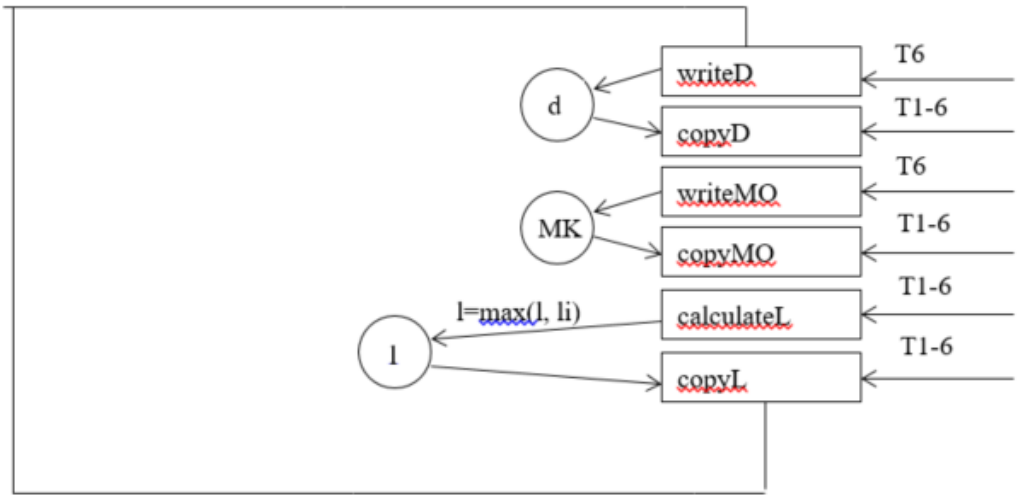


Рис 3.1 - Структурна схема монітора задачі взаємного виключення завд. 2.

На основі алгоритму процесів розроблені структурні схеми моніторів (рис. 2.2, 2.3), що реалізовані з допомогою synchronized-методів мови програмування Java.

Для взаємодії процесів використовується два класи:

1) *ResourceMonitor* – призначений для вирішення задачі взаємного виключення.
2) *SynchronizeMonitor* – призначений для вирішення задачі синхронізації.

Клас *ResourceMonitor* має функції, що дозволяють лише копіювати спільний ресурс:

- *CopyL* – копіювання спільного ресурсу – числа *l*.
- *CopyD* – копіювання спільного ресурсу – числа *d*.
- *CopyMK* – копіювання спільного ресурсу – матриці *MK*.

Методи ResourceMonitor призначені для зміни спільного ресурсу:

- *WriteMK* – запис матриці *MK* в монітор.
- *WriteD* – запис числа *d* в монітор.
- *CalculateL* – виконання операції $l = max(l, l_i)$.

Методи монітора *SynchronizeMonitor* призначені для посилання сигналу:

- *SignalInput* – сигнал про закінчення вводу.
- *SignalCalculate1_2* – сигнал про закінчення обчислення l_i та l.
- *SignalCalculate3* – сигнал про закінчення обчислення MA_H.

Методи монітора *SynchronizeMonitor* призначені для очікування на завершення подій:

- *WaitInput* – очікування завершення вводу.
- *WaitCalculate1_2* – очікування завершення обчислення l_i та l.
- *WaitCalculate3* – очікування завершення обчислення MA_H.

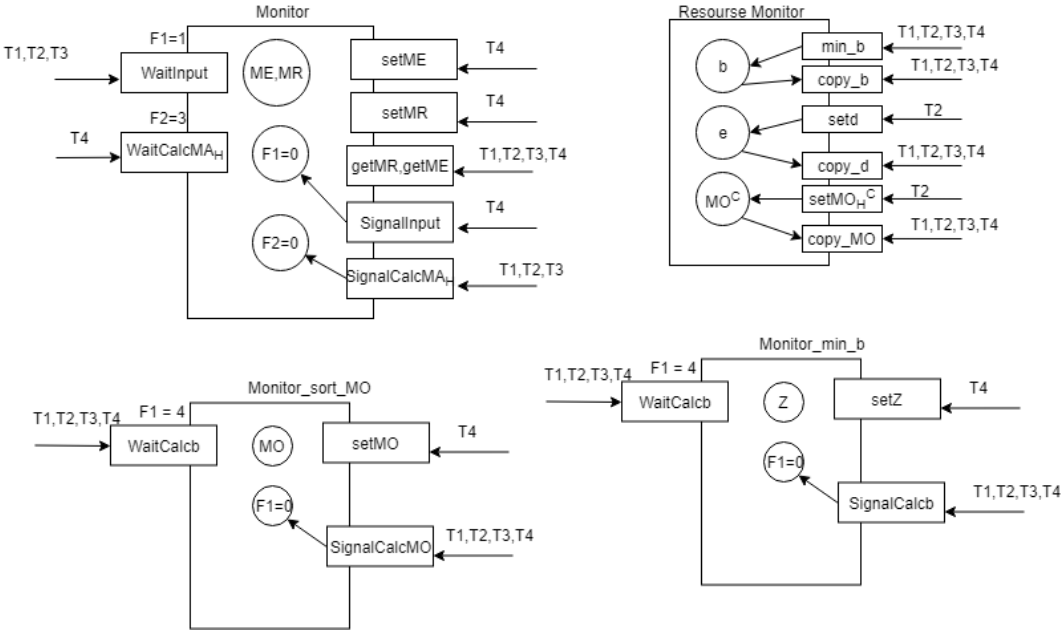


Рис. 3.2 Схема взаємодії задач для системи зі спільною пам’яттю завд.1

3.2.4 Розробка програми

Програма для системи з спільною пам'яттю реалізована на мові Java, і складається з класів Main, Utils (який містить процедури для математичних обчислень), Monitor,Monitor_sort, Monitor_min_b та Resourse_Monitor.

P – кількість потоків для виконання.

Для взаємодії між задачами використовуються монітори. Класс Monitor використовується для синхронізації потоків.

Для забезпечення виконання задачі взаємного виключення використовуються критичні секції, що забороняють одночасний доступ до спільного ресурсу.

В даному випадку ці дії виконує класс Resourse_Monitor .

Класс Monitor_min_b використовується для пошуку мінімуму.

Класс Monitor_sort використовується для сортування.

3.3. Тестування програмного забезпечення структури з спільною пам’яттю

.Тому при тестуванні послідовно використовуються 1, 2, 3, 4 процесорів, для яких визначаються час виконання програми . При цьому встановлюється декілька значень розмірності векторів (матриць) N = 1500, 2000, 2500, 3000. В таблиці 2.1 відображаються значення часу для різних N та P.

Таблиця 3.3. Час виконання програми з спільною пам'яттю (значення в секундах)

N|P	T1	T2	T3	T4
1500	62.684	33.990	23.193	16.645
2000	124.726	68.773	44.472	32.249
2500	227.053	123.552	81.588	59.179
3000	452.602	248.650	160.603	120.413

Підрахунок коефіцієнту прискорення (КП) виконується за формулою

$КП = T1 / T_p$

Таблиця 3.4. Значення КП для програми з спільною пам'яттю

N	Кількість процесорів(P)			
	1	2	3	4
1500	1	1.8441	2.7027	3.765
2000	1	1.8135	2.8045	3.8675
2500	1	1.8377	2.7839	3.8367
3000	1	1.8202	2.8181	3.7587

Підрахунок коефіцієнту ефективності (КЕ) відбувається за формулою

КЕ = КУ / Р * 100%

Таблиця 3.5. Значення КЕ для програми з спільною пам'яттю

N	Кількість процесорів (P)			
	1	2	3	4
1500	100	92.20	90.09	94.12
2000	100	90.67	93.48	96.68
2500	100	91.88	92.79	95.91
3000	100	91.01	93.93	93.96

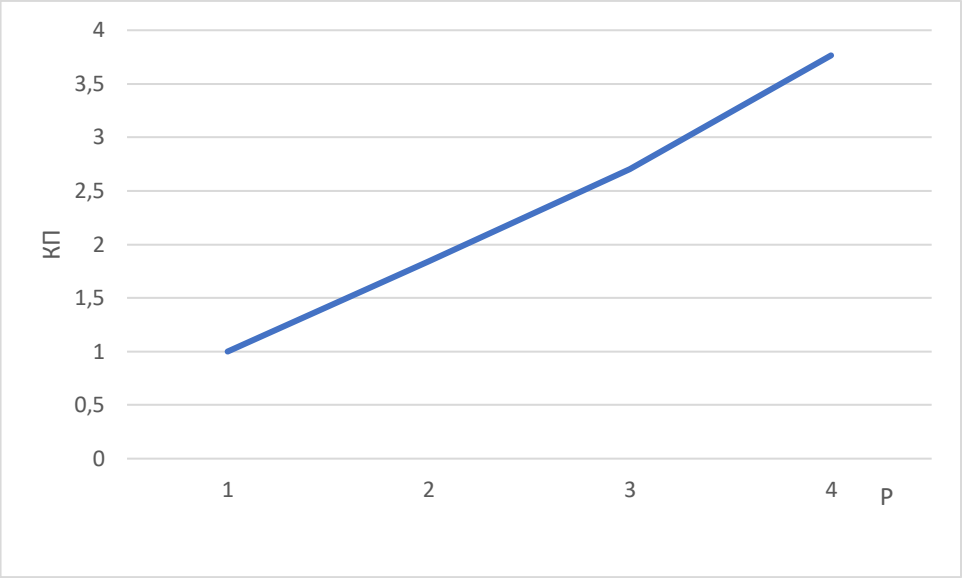


Рис 3.3. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 1500

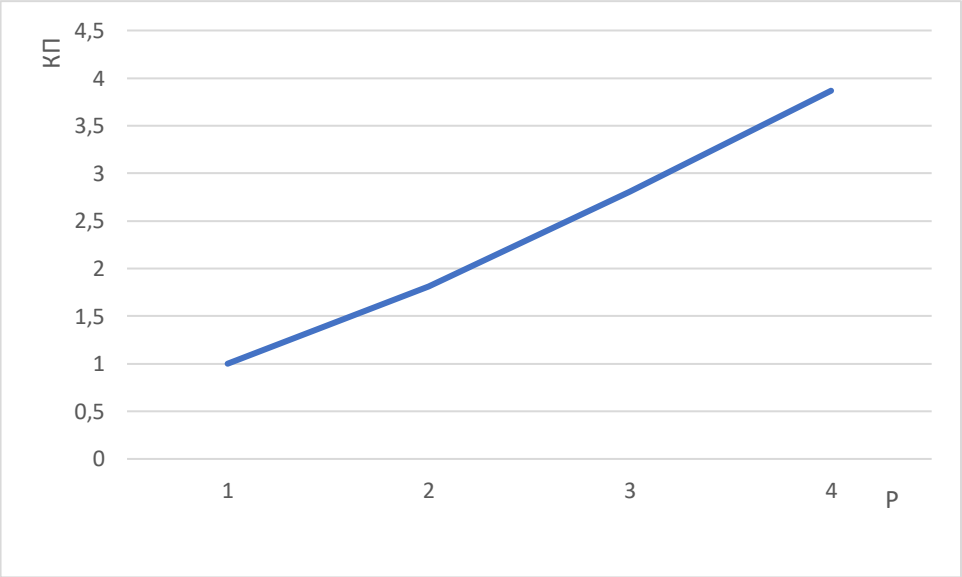


Рис 3.4. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 2000

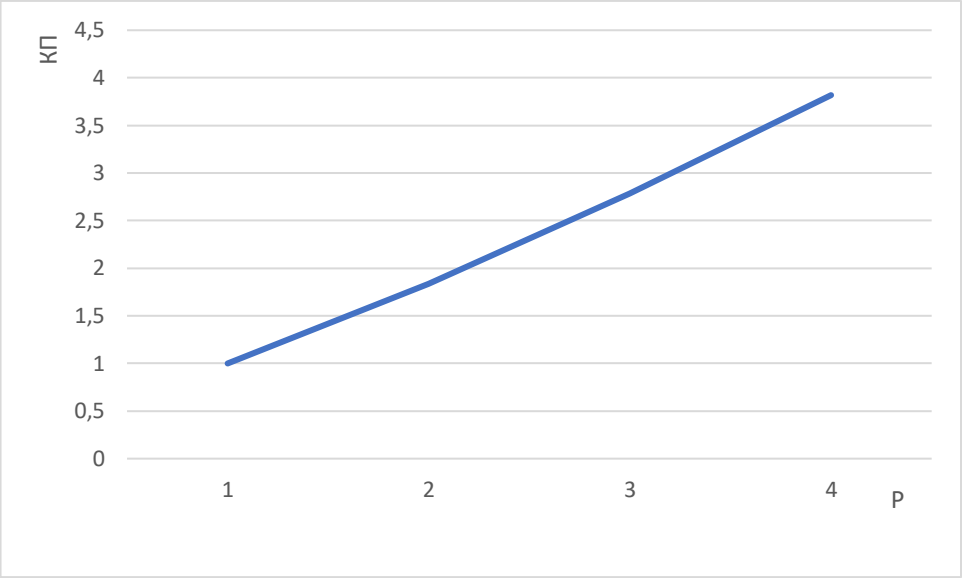


Рис 3.5. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 2500

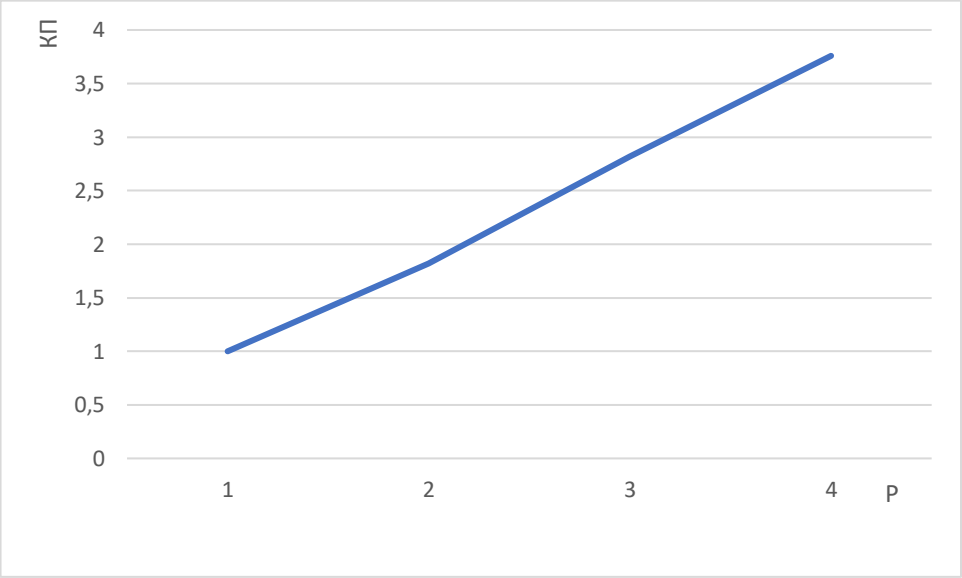


Рис 3.6. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 3000

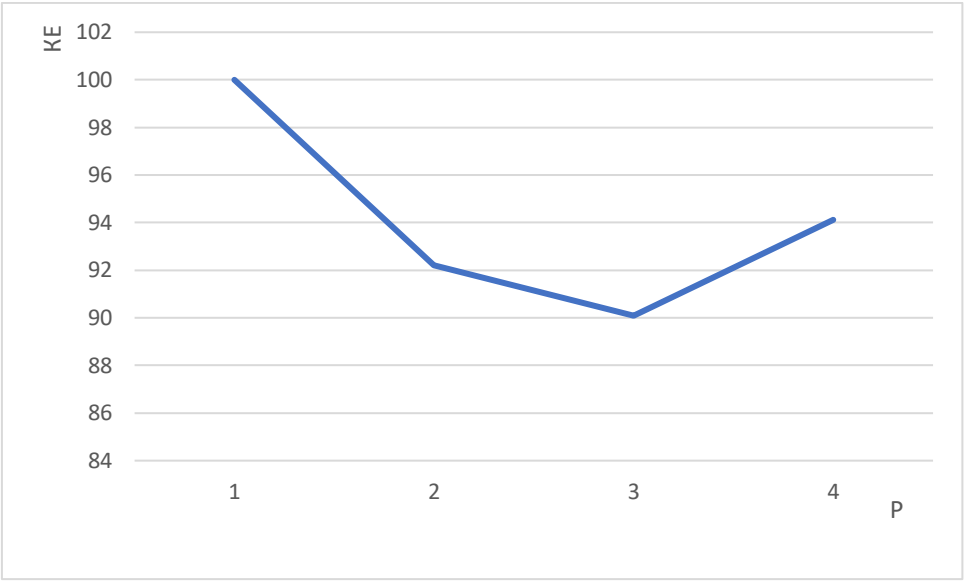


Рис 3.7. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 1500

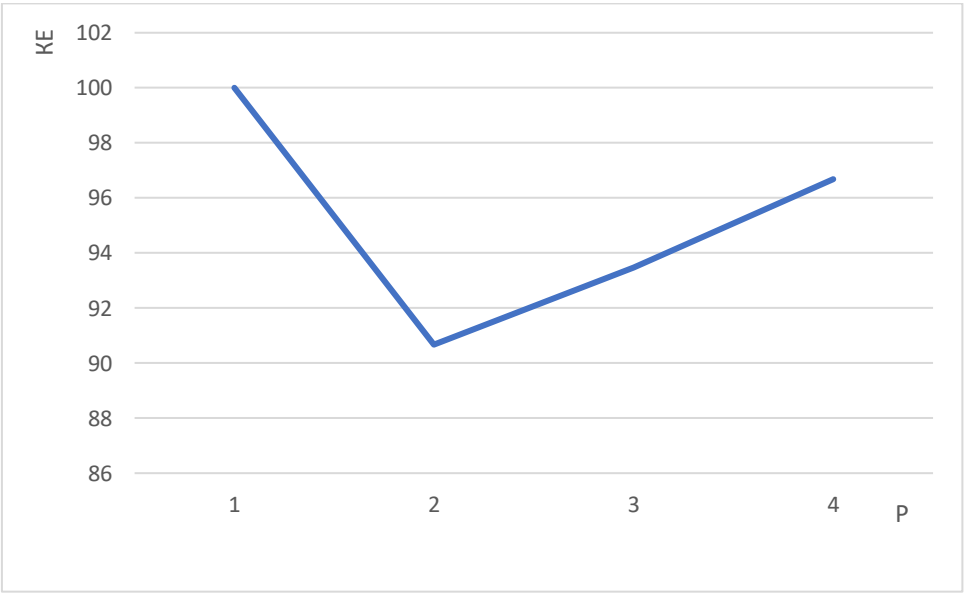


Рис 3.8. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 2000

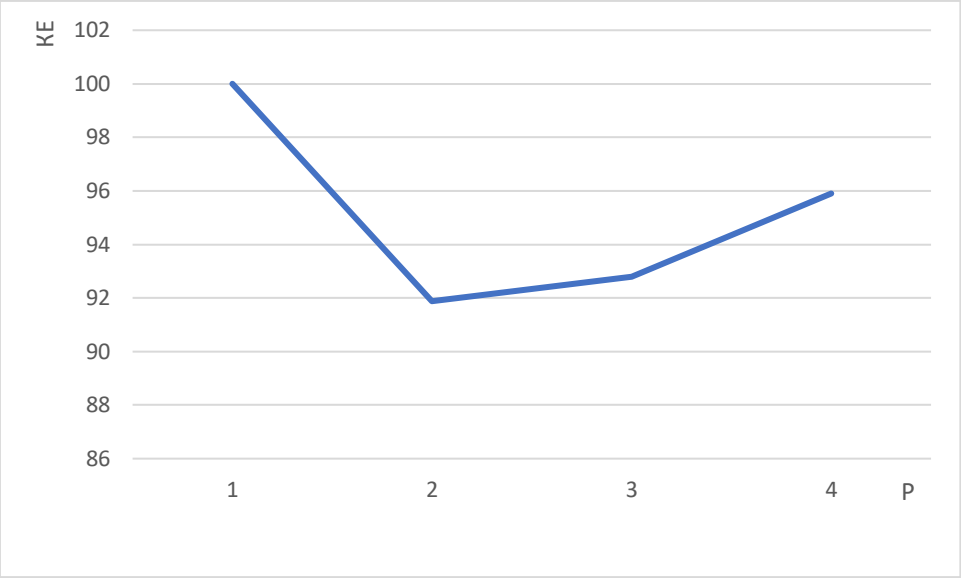


Рис 3.9. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 2500

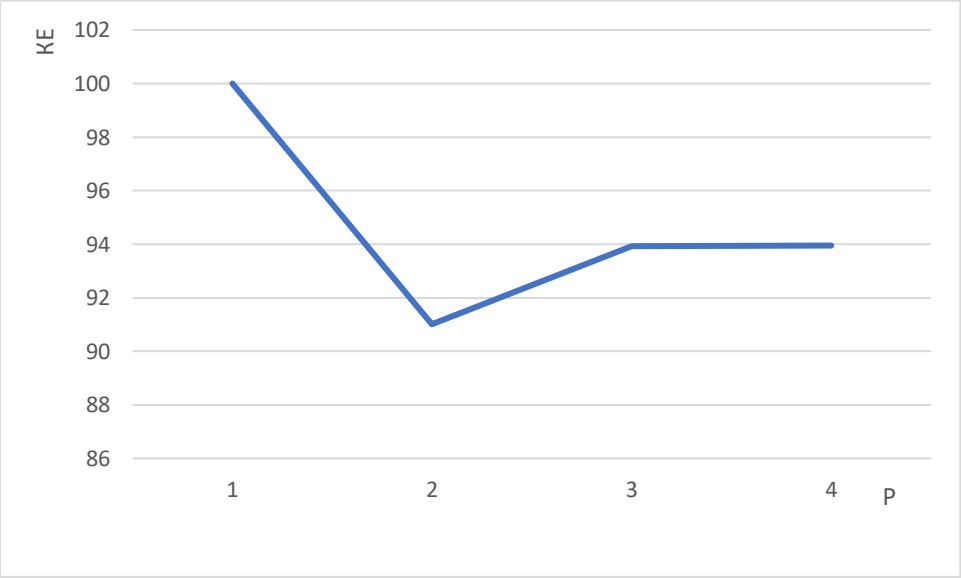


Рис 3.10. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 3000

Висновок до розділу 3

У даному розділі було зібрано апаратну частину, та розроблено програмне забезпечення на основі отриманих результатів з розділів 1 та 2.Було обрано процесор Threadripper 1950x та мова програмування Java.

Було протестовано систему на прикладі двох задач лінійної алгебри. За результатами яких було визначено залежність коефіцієнту ефективності від кількості процесорів. Також було визначено коефіцієнт прискорення при збільшенні кількості потоків. На основі алгоритму процесів розроблені структурні схеми моніторів, що реалізовані з допомогою synchronized-методів мови програмування Java.

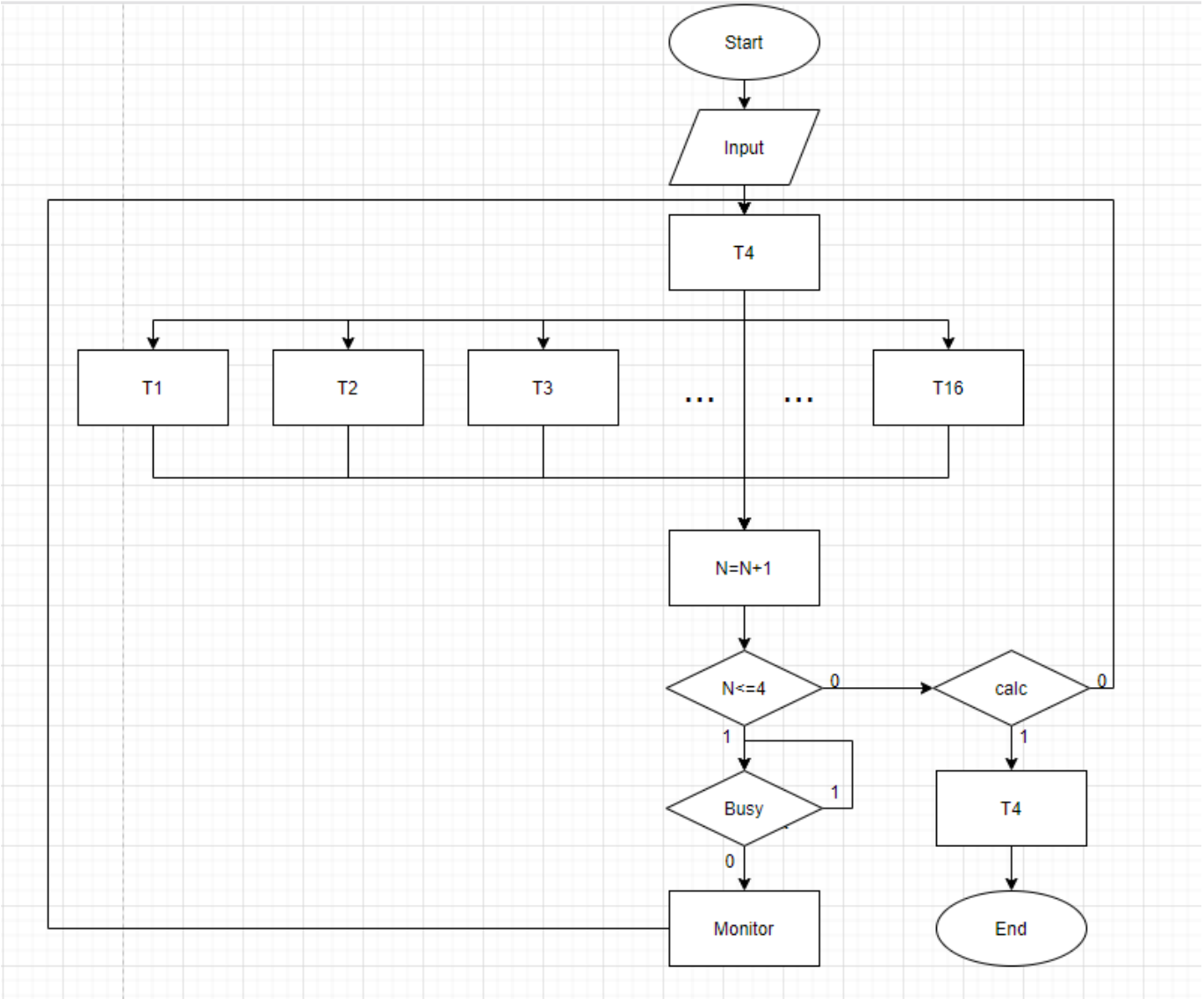
ВИСНОВКИ

1. У представленій роботі було розглянуті процесори та мови і бібліотеки програмування котрі підтримують багатопотоковість .
2. На основі даних було зібрано програмно-апаратний комплекс з основним нахилом на відношення ціни до якості.
3. Було проведено моделювання роботи програми та аналіз її результатів, яке показало, що використання більшої кількості ядер в багатопотокових обчисленнях сприяє швидшому обрахунку задач лінійної алгебри більшої складності.

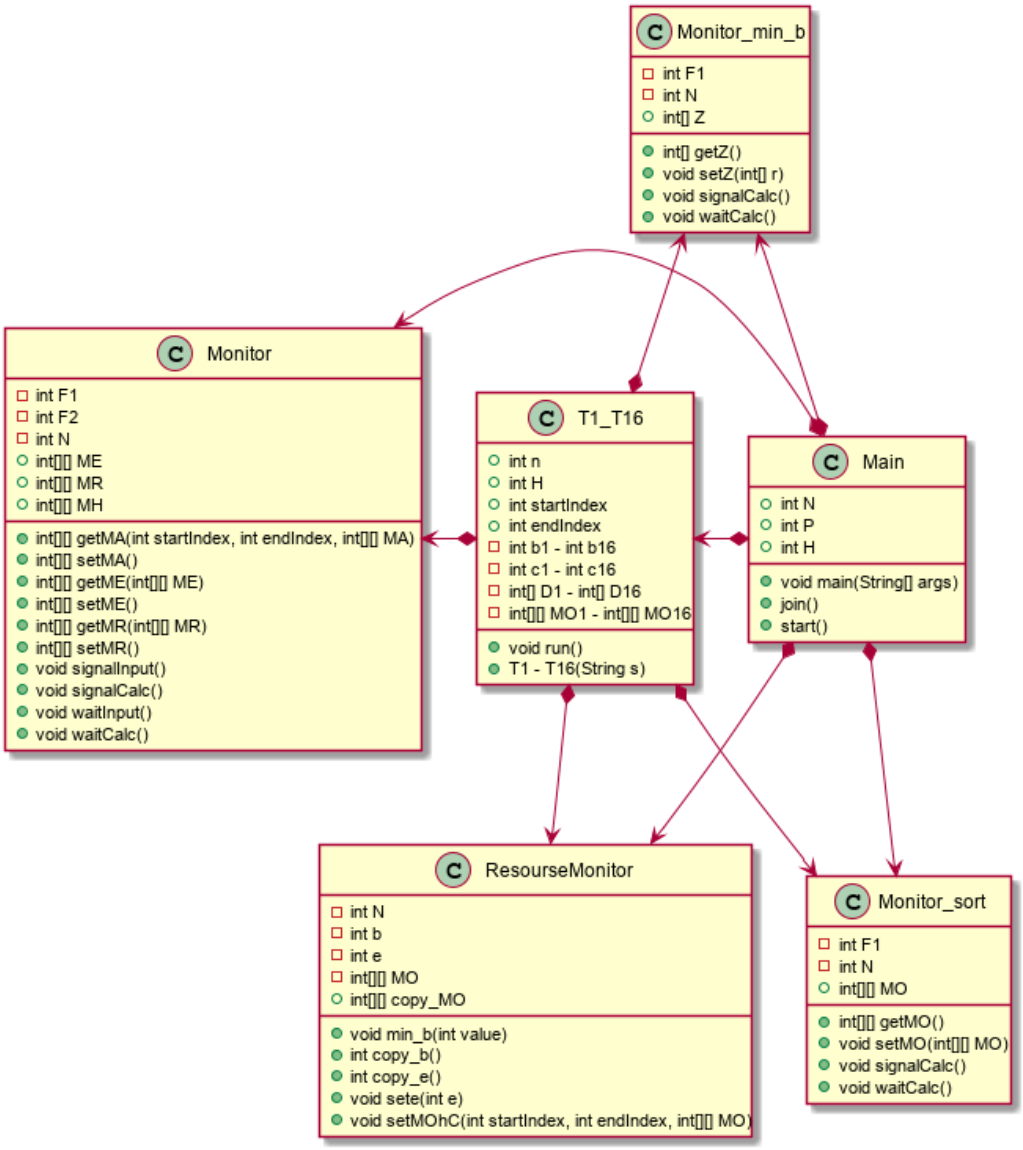
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Огляд процесора i7-6950X [Електронний ресурс] – Режим доступу до ресурсу: https://3dnews.ru/933795.
2. Огляд процесора Intel Core i9-9900X [Електронний ресурс] – Режим доступу до ресурсу: https://3dnews.ru/984758/obzor-core-i9-9900x.
3. Огляд процесора Intel Core i9-9800X [Електронний ресурс] – Режим доступу до ресурсу: https://ark.intel.com/content/www/ru/ru/ark/products/189122/intel-core-i7-9800x-x-series-processor-16-5m-cache-up-to-4-50-ghz.html.
4. Огляд процесора Intel Core i9-9900X [Електронний ресурс] – Режим доступу до ресурсу: 16. https://www.intel.ru/content/www/ru/ru/products/processors/core/x-series/i9-9900x.html.
5. Огляд процесора Intel Core i9-9900X [Електронний ресурс] – Режим доступу до ресурсу: 17. https://benchmarkdb.ru/cpu/intel/core_i9-9900x/.
6. Огляд процесора Intel Core i9-9900X [Електронний ресурс] – Режим доступу до ресурсу: https://askgeek.io/ru/cpus/Intel/Core-i9-9900X.
7. Огляд процесора Intel Core i9-9900X [Електронний ресурс] – Режим доступу до ресурсу: https://www.hardwareluxx.ru/index.php/news/hardware/prozessoren/45685-intel-benchmarks-i9-9900k-i9-9980xe-i9-9900x.html.
8. Огляд процесора Intel Core i9-10980X [Електронний ресурс] – Режим доступу до ресурсу: https://3dnews.ru/998360/obzor-intel-core-i910980xe.
9. Огляд процесора Intel Core i9-10980X [Електронний ресурс] – Режим доступу до ресурсу: https://www.ixbt.com/platform/intel-core-i9-10980xe-cascade-lake-test.html.
10. Огляд процесора Intel Core i9-10980X [Електронний ресурс] – Режим доступу до ресурсу: https://www.hardwareluxx.ru/index.php/artikel/hardware/prozessoren/48484-test-i-obzor-intel-core-i9-10980xe-18-yadernyj-protsessor-dlya-rabochikh-stantsij.html?start=12.
11. Огляд процесора AMD R9 3950 [Електронний ресурс] – Режим доступу до ресурсу: https://3dnews.ru/997670/obzor-amd-ryzen-9-3950x.
12. Огляд процесора AMD R9 3950 [Електронний ресурс] – Режим доступу до ресурсу: https://www.overclockers.ua/cpu/amd-ryzen-9-3900x/all/.
13. Огляд процесора AMD R9 3950 [Електронний ресурс] – Режим доступу до ресурсу: https://www.amd.com/ru/products/cpu/amd-ryzen-9-3950x.
14. Огляд процесора AMD R9 3950 [Електронний ресурс] – Режим доступу до ресурсу: https://3dnews.ru/997096/obzor-amd-ryzen.
15. Огляд процесора AMD R9 3900 [Електронний ресурс] – Режим доступу до ресурсу: https://itc.ua/articles/obzor -amd.
16. Огляд процесора AMD TR 1950 [Електронний ресурс] – Режим доступу до ресурсу: 28. https://www.amd.com/ru/products/ryzen-threadripper.

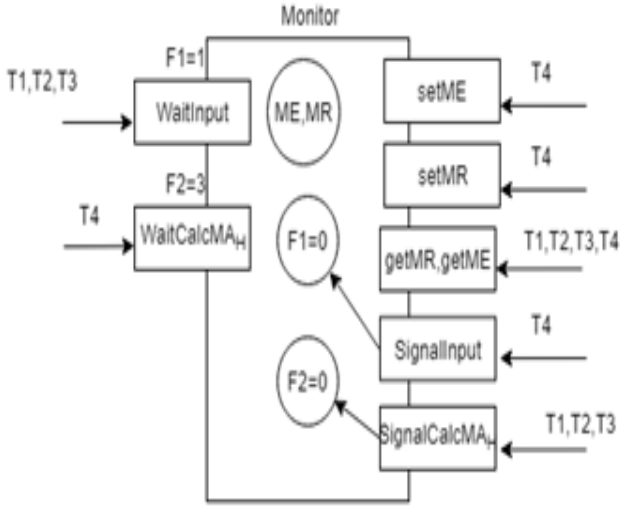
17. Огляд процесора AMD TR 2950 [Електронний ресурс] – Режим доступу до ресурсу: https://www.ixbt.com/platform/amd-ryzen-threadripper-2950x-2990wx-review.html.
18. Огляд процесора AMD R9 3900 [Електронний ресурс] – Режим доступу до ресурсу: https://www.amd.com/ru/products/cpu/amd-ryzen-9-3900x.
19. Огляд процесора AMD TR 3970 [Електронний ресурс] – Режим доступу до ресурсу: https://3dnews.ru/1002791/obzor-amd-ryzen-threadripper-3970x-3960x.
20. Огляд процесора AMD R9 3950X [Електронний ресурс] – Режим доступу до ресурсу: https://3dnews.ru/997670/obzor-amd-ryzen-9-3950x.
21. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. Посібник / І. А. Жуков, О. В. Корочкін, 2005. – 226 с.
22. Введенння в багатопотоковість [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/tutorial/11.1.php.
23. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. Посібник / І. А. Жуков, О. В. Корочкін, 2005. – 226 с.
24. Процеси та потоки в Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/developerworks/ru/library/l-python_part_9/index.html.
25. Паралельне програмування на основі MPI [Електронний ресурс] – Режим доступу до ресурсу: http://www.hpcc.unn.ru/mskurs/RUS/DOC/ppr04.pdf.
26. Реалізація багатопотоковості в WinAPI [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/cpp/parallel/multithreading-with-c-and-win32?view=vs-2019.
27. Паралельне програмування на OpenMP [Електронний ресурс] – Режим доступу до ресурсу: http://ccfit.nsu.ru/arom/data/openmp.pdf.

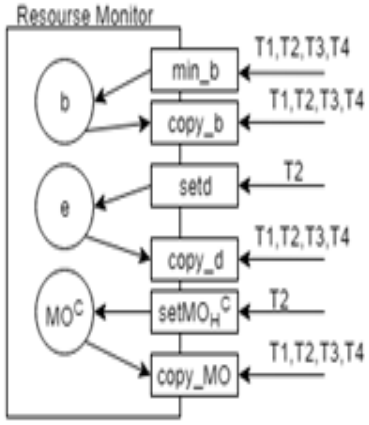


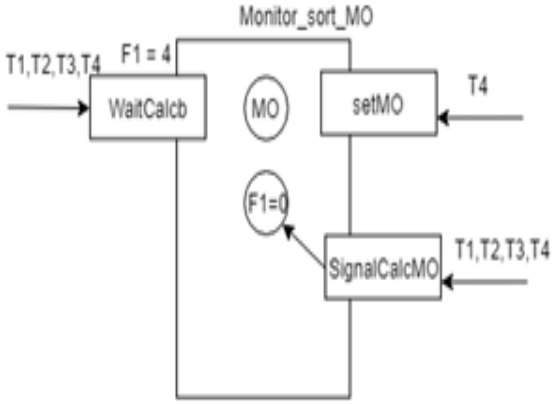
					ДП.4665.04.000 А1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		*Осіпов І.О.*			*Принципова схема алгоритму*	Літ.	Аркуш 1 / Аркушів 1
Перевірив		*Корочкін О.В.*					
Реценз.						*НТУУ «КПІ», ФІОТ, ІО-62*	
Н. Контр.		*Сімоненко В.П.*					
Затвердив							

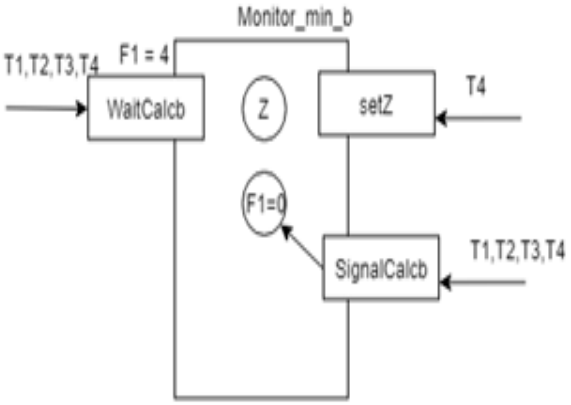


					ДП.4665.05.000 А2		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		*Осіпов І.О.*			*Функціональна схема*	Літ.	Аркуш 1 / Аркушів 1
Перевірив		*Корочкін О.В.*					
Реценз.						*НТУУ «КПІ», ФІОТ, ІО-62*	
Н. Контр.		*Сімоненко В.П.*					
Затвердив							









					ДП.4665.06.000 А3		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		*Осіпов І.О.*			*Структурна схема*	Літ.	Аркуш 1 / Аркушів 1
Перевірив		*Корочкін О.В.*					
Реценз.						*НТУУ «КПІ», ФІОТ, ІО-62*	
Н. Контр.		*Сімоненко В.П.*					
Затвердив							